Epsilon User Manual-v2.3

Idea Device Support
support@ideadevice.com

©*Copyright 2010 Idea Device Technologies*

Contents

1 Getting Started **7**

1.1 Login . . . 7
1.2 Dashboard . . . 7
1.2.1 Global Tabs . . . 9
1.2.2 Dashboard Tables . . . 9
1.2.3 Shortcut Buttons . . . 9
1.3 Creating a group and a user . . . 9
1.4 Setting the default group . . . 11
1.5 Create a Machine . . . 11
1.6 Create Credentials . . . 13
1.7 Create Properties . . . 13
1.8 Create Entities . . . 15
1.9 Create a Workflow . . . 17
1.10 Using EMC Macros, Input task, Loop task, call workflow task . . . 23
1.11 Scheduler . . . 27
1.12 View the run logs . . . 28
1.13 Abort, Rollback, Re Run a workflow . . . 30
1.14 Workflow Edit Page . . . 31
1.14.1 clone a workflow . . . 32
1.14.2 Version Control . . . 33
1.14.3 Inactivating a Workflow . . . 34
1.14.4 Setting Pre conditions for a workflow run . . . 34
1.14.5 Generate a process document . . . 34
1.14.6 Setting a workflow as Self Service Application . . . 35

2 Tasks **39**

2.1 Run Shell . . . 39
2.1.1 Passing arguments to a script . . . 40
2.2 Call workflow . . . 40
2.3 Input . . . 40
2.4 Interactive . . . 43
2.5 Approval . . . 44
2.6 Decision . . . 46
2.7 Parallel . . . 46
2.8 Meta . . . 47
2.9 Notification . . . 47

2.10 Email . . . 48
2.11 Manual . . . 49
2.12 Delay . . . 49
2.13 Eval . . . 49
2.14 Dynamic . . . 49
2.15 Update . . . 50
2.15.1 Rollback for Update task . . . 50
2.16 Confirm . . . 50
2.17 Report . . . 50
2.18 Loop . . . 50
2.18.1 While loop . . . 50
2.18.2 For Loop . . . 51
2.18.3 Loop Over Multiple Machines . . . 51
2.19 Method to Succeed or Fail a task . . . 52

3 Properties **55**
3.1 Property Inheritance . . . 55

4 Macros **57**
4.1 Macros to access the output of previous tasks . . . 58
4.2 Query Entity, Machines and Credentials . . . 58
4.3 Nested Macro Evaluation . . . 63

5 Epsilon API **65**
5.1 LDAP APIs for Microsoft AD Integration . . . 65
5.1.1 AD Classes . . . 65
5.2 List of LDAP APIs . . . 69
5.3 APIs for Resuming Workflows . . . 71
5.4 API to Remote Copy a File . . . 72
5.5 JDBC APIs . . . 72
5.6 Date-Time Manipulation in Epsilon Scripts . . . 74

6 S-Expression Language **75**
6.1 Introduction . . . 75
6.2 Syntax . . . 75
6.2.1 Data Types . . . 75
6.2.2 Execution . . . 76
6.3 Built In Operators . . . 76
6.3.1 + (addition) . . . 76
6.3.2 - (subtraction) . . . 76
6.3.3 * (multiplication) . . . 77
6.3.4 / (division) . . . 77
6.3.5 > (greater than) . . . 77
6.3.6 < (less than) . . . 77
6.3.7 >= (greater than or equal to) . . . 78
6.3.8 <= (less than or equal to) . . . 78
6.3.9 = (equality) . . . 78
6.3.10 equal? . . . 78
6.3.11 and? . . . 79
6.3.12 or? . . . 79

6.3.13 true? . . . 79
6.3.14 false? . . . 80
6.3.15 quote . . . 80
6.3.16 list . . . 80
6.3.17 list? . . . 80
6.3.18 if . . . 81
6.3.19 define . . . 81
6.3.20 set! . . . 82
6.3.21 lambda . . . 82
6.3.22 function . . . 83
6.3.23 fn . . . 83
6.3.24 begin . . . 83
6.3.25 length . . . 83
6.3.26 integer . . . 83
6.3.27 number . . . 84
6.3.28 string . . . 84
6.3.29 split . . . 84
6.3.30 join . . . 84
6.3.31 concat . . . 85
6.3.32 first . . . 85
6.3.33 car . . . 85
6.3.34 rest . . . 85
6.3.35 cdr . . . 86
6.3.36 range . . . 86
6.3.37 map . . . 86
6.3.38 reduce . . . 87
6.3.39 regexp-search . . . 87
6.3.40 regexp-substitute . . . 87
6.3.41 regexp-find-all . . . 87
6.3.42 regexp-match-group . . . 87
6.3.43 regexp-match-groups . . . 88
6.3.44 get-property . . . 88
6.3.45 get-macro . . . 88
6.3.46 filter-macro . . . 88
6.3.47 filter-macro-lines . . . 88

7 Storage and Type Interface **89**
7.1 Introduction . . . 89
7.2 Basic Types . . . 89
7.2.1 String . . . 89
7.2.2 Number . . . 89
7.2.3 Secret . . . 89
7.2.4 Date . . . 89
7.2.5 Boolean . . . 89
7.2.6 Float . . . 89
7.3 User Defined Types . . . 90

Appendices **93**
.1 Builtin Macro Listing . . . 95

Chapter 1

Getting Started

1.1 Login

Your Epsilon administrator should have created an Epsilon account and communicated the account details to you. The account details comprise of the following attributes:

1. Username
2. Password
3. Epsilon server address

The *groups* you are member of are another important attribute. A group would have been assigned to you by the Epsilon administrator (user `admin`) when your account was created. Every user is member of at least one group.

Epsilon is accessed through a browser interface. To login enter the server address (e.g. http://epsilon.example.com) into your browser. Epsilon will present a login page for you.

A screenshot of the login page is shown in section 1.1.

Enter your username and password in the username and password text fields and click on the Login button. You can also press the enter key. If your username and password are not successfully matched with a valid user you are returned to the login screen.

1.2 Dashboard

After you login you are shown the dashboard page. The dashboard page provides a concise overview of activities of your user and group. It's components automatically refresh every few seconds to show the latest data.

In Figure 1.2 you can see a dashboard screenshot. The currently logged in user and group are displayed in the top right corner of the screen. The group name display also works as a selector to change your current group (if you are member of more than one group).

The top menu also contains the Logout, Preferences, Feedback, Support and FAQ links. The top menu is available globally in the Epsilon application.

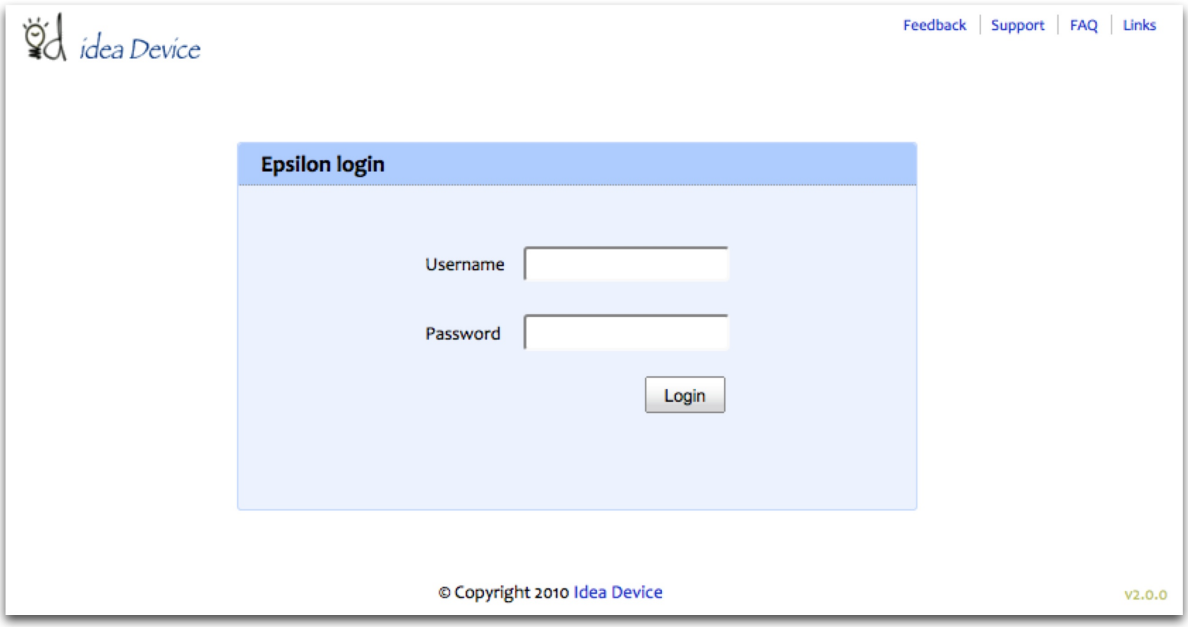

Figure 1.1: Epsilon Login Page

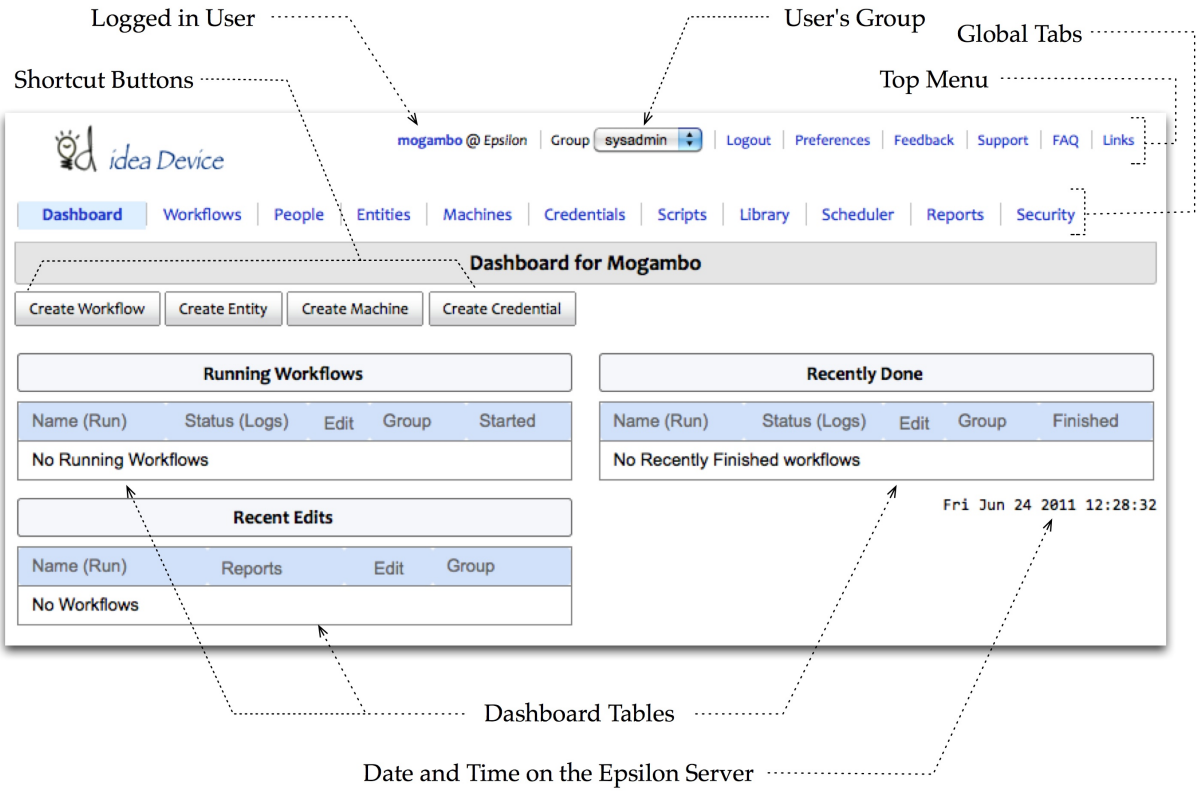

Figure 1.2: Dashboard

1.2.1 Global Tabs

Under the top menu are the global tabs. The global tabs allow you to access different functional areas of the application. For added flexibility you can simultaneously open several global tabs in browser windows. Almost all links in Epsilon behave in a similar manner.

1.2.2 Dashboard Tables

The dashboard page uses multiple tables to display the current state of your workflows. Usually you see the following three tables.

Running Workflows

This table displays the currently running workflows for your group. As workflows finish running they move to the Recently Done table.

Recently Done

This table displays the latest 10 workflows that have recently finished running.

Recent Edits

This table shows the latest 10 workflows edited.

1.2.3 Shortcut Buttons

The shortcut buttons bar on the upper left side of the page allows you to jump to commonly used functions with one click. The four shortcuts are:

1. Create a Workflow
2. Create an Entity
3. Create a Machine
4. Create a Credential

1.3 Creating a group and a user

- Group: A collection of users within Epsilon. Group is the owner of all other metadata within Epsilon. Group within Epsilon can model real world teams or can represent a virtual team made up of users with similar functionality. Eg: dba, sr mgmt, vmware operators
- User: A named user within Epsilon. Users login with a user account and belong to one or more Group. Eg: alok, jayarama, madhulika

By default Epsilon comes seeded with a user admin and a group wheel. The following steps will illustrate how to create a new group and a user and associate the user to the new group.

1. Log in to Epsilon using the following credentials Username: admin Password: adm1n
2. Click on the tab People and click create New group button (figure 1.3)

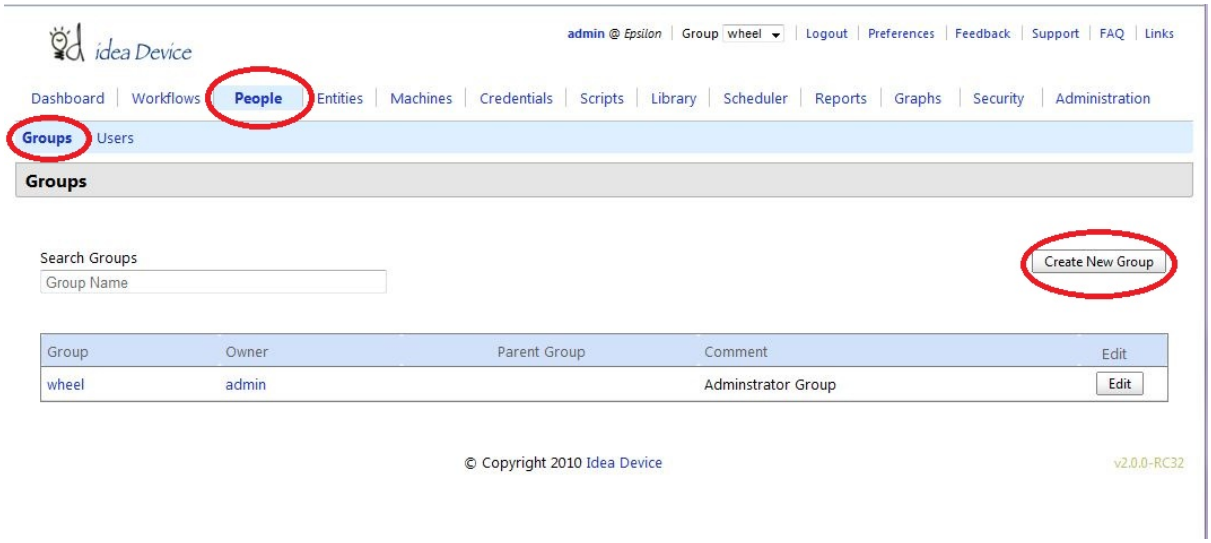

Figure 1.3: Create a New Group

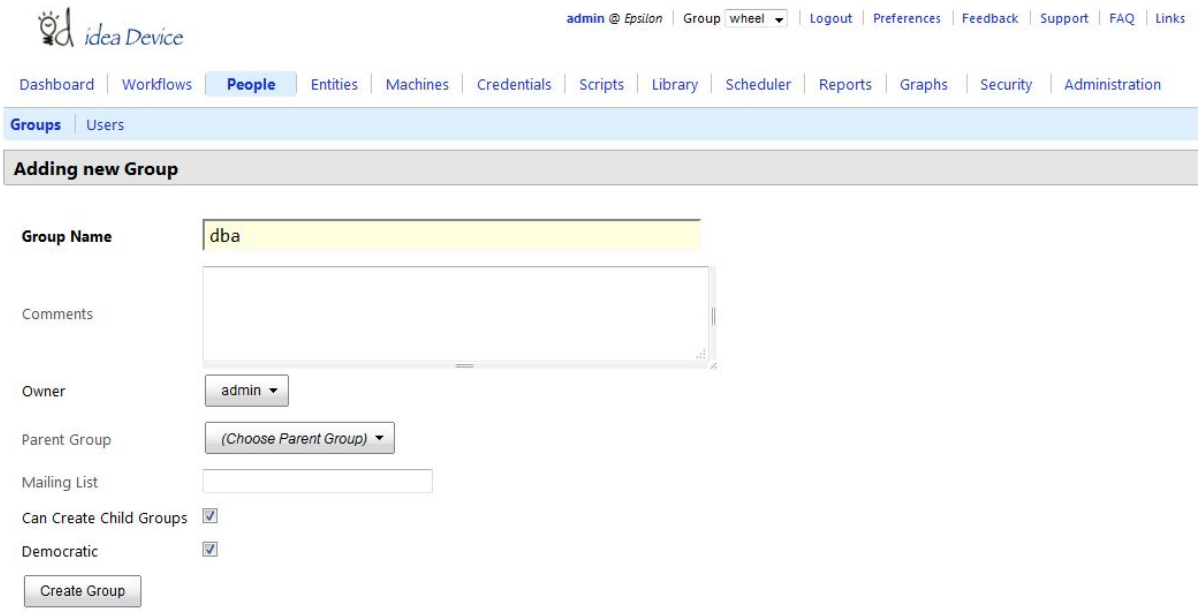
idea Device
admin @ Epsilon | Group wheel | Logout | Preferences | Feedback | Support | FAQ | Links
Dashboard | Workflows | People | Entities | Machines | Credentials | Scripts | Library | Scheduler | Reports | Graphs | Security | Administration
Groups | Users
Adding new Group
Group Name
dba
Comments
Owner
admin
Parent Group
(Choose Parent Group)
Mailing List
Can Create Child Groups
Democratic
Create Group

Figure 1.4: Create a New Group

3. Enter the group name, choose the owner as admin and press create group (figure 1.4)

4. To create a user click on the Users tab and press the button Create New User (figure 1.5)

5. Enter the details and press create user. The password will be mailed to the user (figure 1.6)

6. Users can be edited to reset the password and other details (figure 1.7)

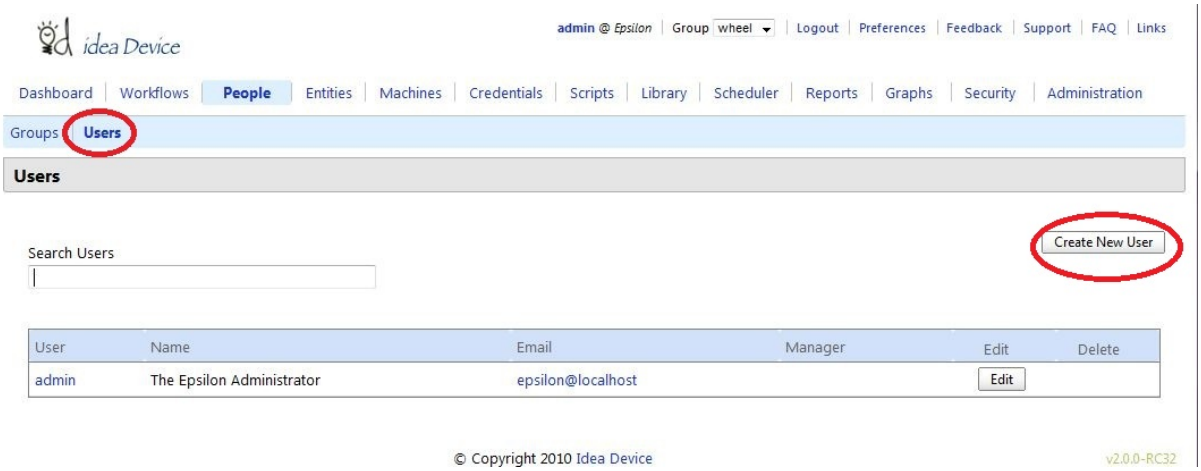
idea Device
admin @ Epsilon | Group wheel | Logout | Preferences | Feedback | Support | FAQ | Links
Dashboard | Workflows | People | Entities | Machines | Credentials | Scripts | Library | Scheduler | Reports | Graphs | Security | Administration
Groups | Users
Users
Search Users
Create New User
User | Name | Email | Manager | Edit | Delete
admin | The Epsilon Administrator | epsilon@localhost | Edit
© Copyright 2010 Idea Device
v2.0.0-RC32

Figure 1.5: Create a New User

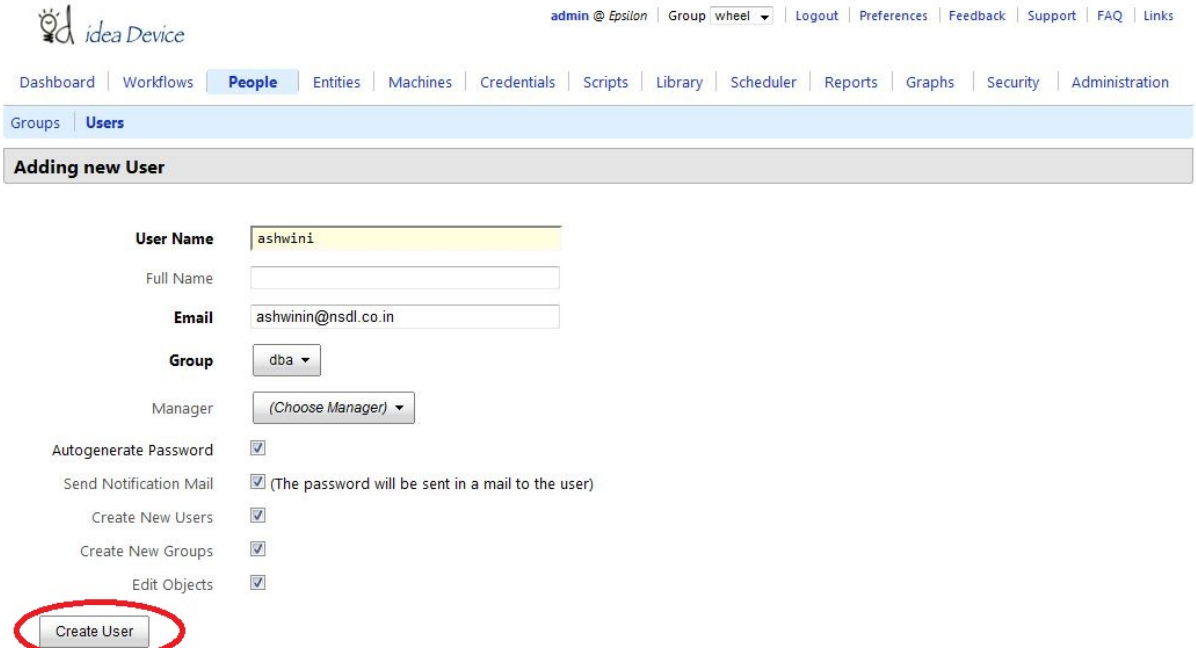
idea Device
admin @ Epsilon | Group wheel | Logout | Preferences | Feedback | Support | FAQ | Links
Dashboard | Workflows | People | Entities | Machines | Credentials | Scripts | Library | Scheduler | Reports | Graphs | Security | Administration
Groups | Users
Adding new User
User Name
ashwini
Full Name
Email
ashwinin@nsdl.co.in
Group
dba
Manager
(Choose Manager)
Autogenerate Password
Send Notification Mail
(The password will be sent in a mail to the user)
Create New Users
Create New Groups
Edit Objects
Create User

Figure 1.6: Create a New User

1.4 Setting the default group

An Epsilon can be a member of multiple groups. The default group can be set using the preference as shown in figure 1.8. Choose the default group from the list of values of groups available to you and press "Save Prefs". Next time you log into the system, the user will belong to this group.

1.5 Create a Machine

Any physical, virtual or logical TCP endpoint node which has to be automated. Epsilon requires that such nodes be registered in the Machines Tab (a builtin CMDB). Every entry in the Machines tab consumes 1 license. Eg: sales db server, VMWare ESX Host, shx02.ideadevice.com.

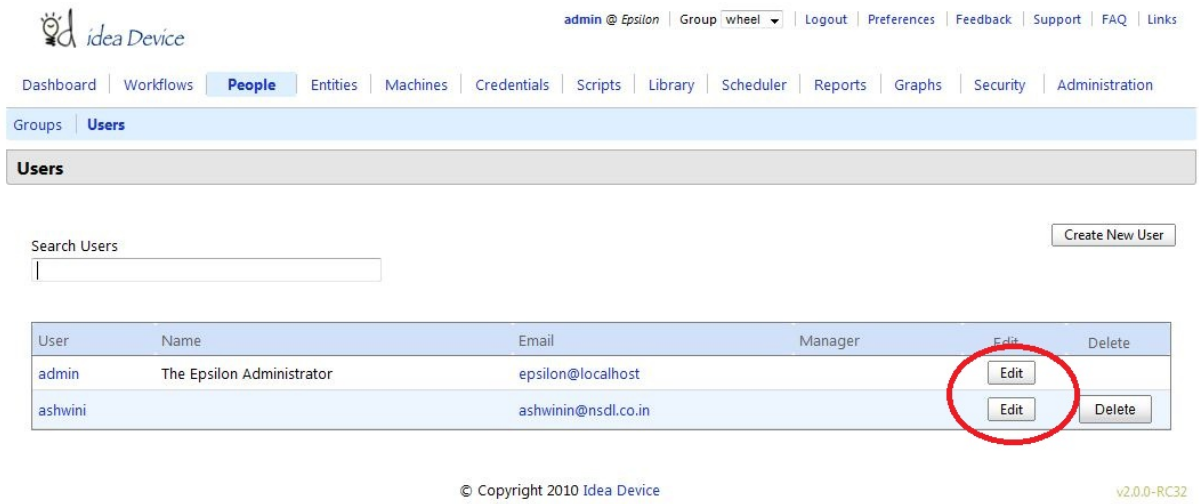
idea Device
admin @ Epsilon | Group wheel | Logout | Preferences | Feedback | Support | FAQ | Links
Dashboard | Workflows | People | Entities | Machines | Credentials | Scripts | Library | Scheduler | Reports | Graphs | Security | Administration
Groups | Users
Users
Search Users
Create New User
User | Name | Email | Manager | Edit | Delete
admin | The Epsilon Administrator | epsilon@localhost | Edit
ashwini | ashwinin@nsdl.co.in | Edit | Delete
© Copyright 2010 Idea Device
v2.0.0-RC32

Figure 1.7: Update a User

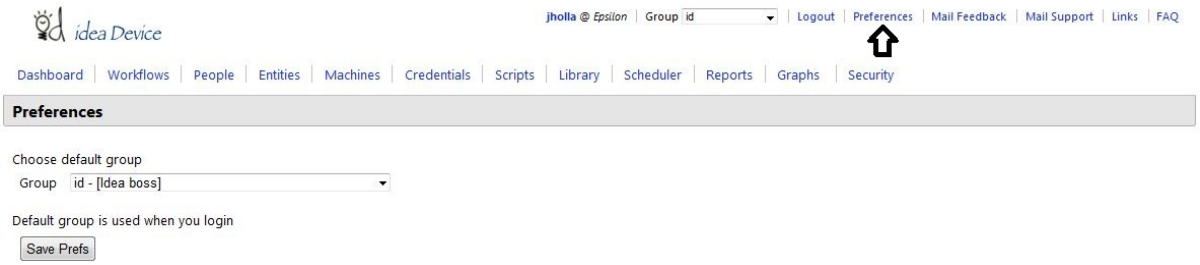
idea Device
jholla @ Epsilon | Group id | Logout | Preferences | Mail Feedback | Mail Support | Links | FAQ
Dashboard | Workflows | People | Entities | Machines | Credentials | Scripts | Library | Scheduler | Reports | Graphs | Security
Preferences
Choose default group
Group id - [Idea boss]
Default group is used when you login
Save Prefs

Figure 1.8: Setting the user preference

Target machines may be of type Linux, Windows, Solaris, HP-UX, AIX, Stratus VOS, Cisco IOS, Juniper-JUNOS or any other device which allows an incoming connection.

- Create a machine by selecting the tab Machines and pressing the button Create New Machine in figure 1.5.

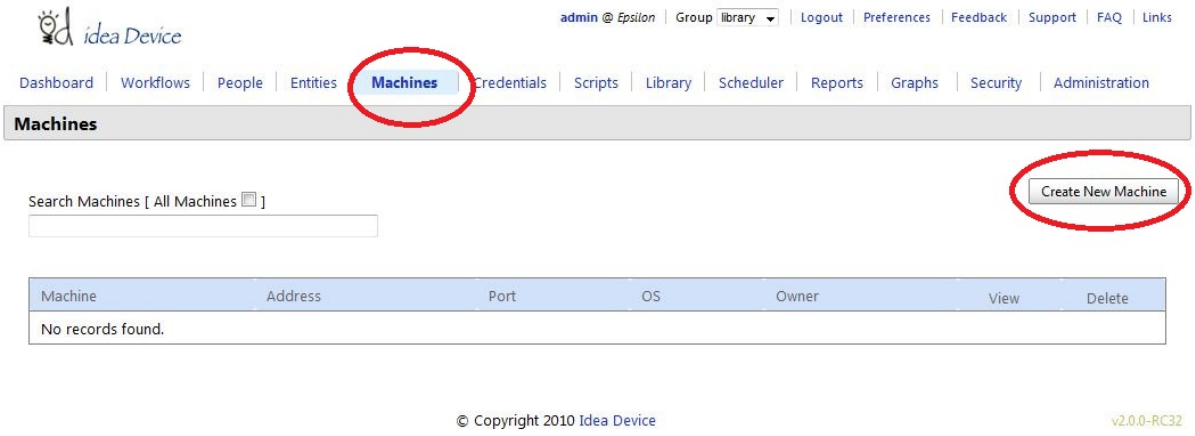
idea Device
admin @ Epsilon | Group library | Logout | Preferences | Feedback | Support | FAQ | Links
Dashboard | Workflows | People | Entities | Machines | Credentials | Scripts | Library | Scheduler | Reports | Graphs | Security | Administration
Machines
Search Machines [All Machines]
Create New Machine
Machine | Address | Port | OS | Owner | View | Delete
No records found.
© Copyright 2010 Idea Device
v2.0.0-RC32

Figure 1.9: create machine

- Enter the Machine Name, IP address or the name and choose the OS, for ssh connections port 22 is the default in figure 1.10.

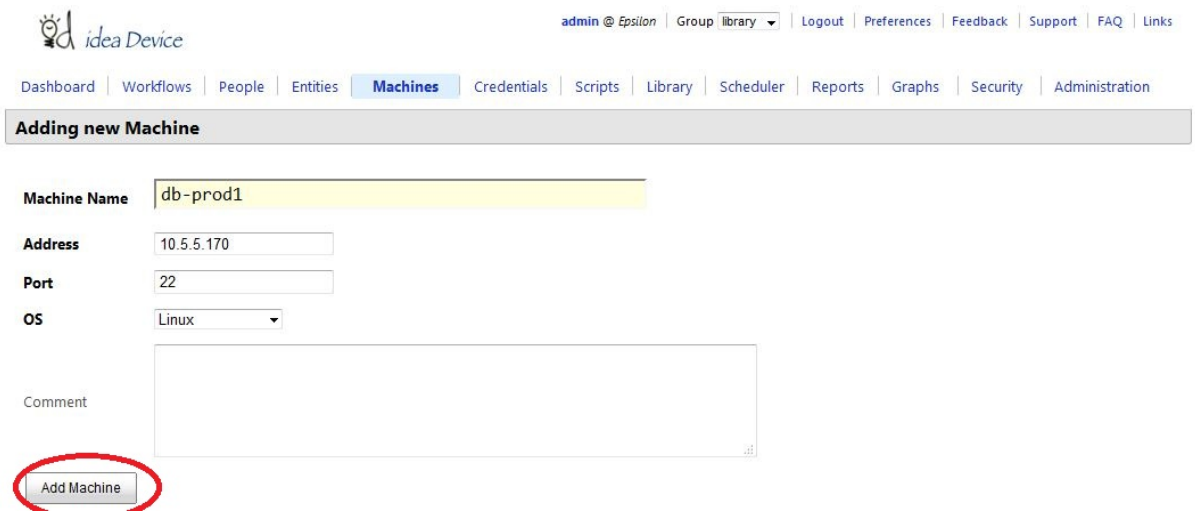

Figure 1.10: Create Machine

1.6 Create Credentials

A valid username/password or username/ssh-key pair. A credential can be used to access 1 machine or more than one machine (in case of generic accounts). A credential is strictly owned by the group which creates the credential and is the only metadata object in Epsilon that cannot be shared. Eg: oracle@sales-db, root, jholla, acme/aaditya.

- Add root credential to db-prod1, by clicking on Credentials tab and pressing the button-Create New Credential (Figure 1.11).

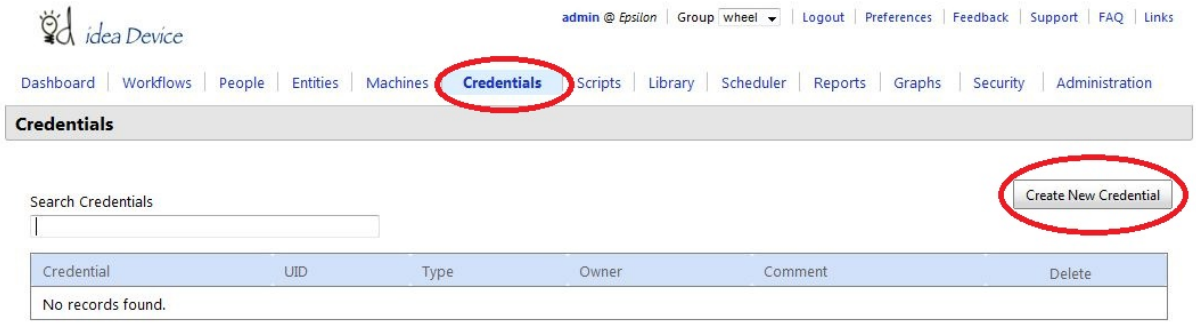

Figure 1.11: Create Credentials

- Enter the user name and pass word or ssh key (figure 1.12)

1.7 Create Properties

A property is a key value pair for eg `ORACLE_SID` =sales-db. The key `@@ORACLE_SID@@` can now be used in scripts, emails, approval messages etc Epsilon will automatically expand

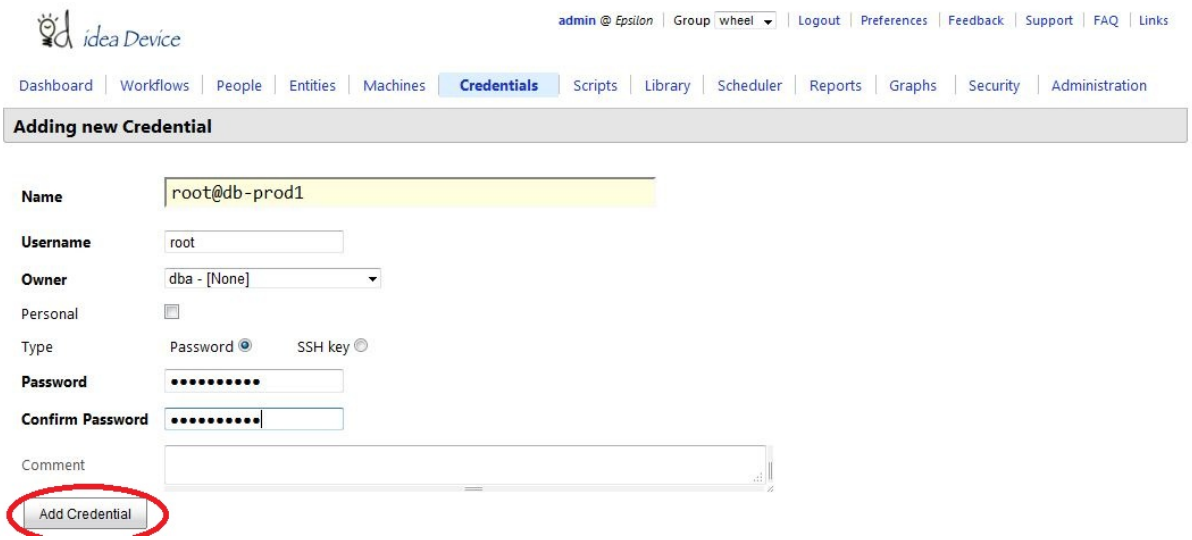

Figure 1.12: Create Credentials

`@@ORACLE_SID@@` to `sales-db` (the @@ tells Epsilon that `ORACLE_SID` must be treated as a key). Properties can be set on Entities, Machines, Groups, Tasks and Workflows. Epsilon figures out the correct expansion of the key by looking at the following objects in-order - Task, Parent Task (if any), Workflow, Machine, Entity and finally Group. The first occurrence is picked up. Properties allow script templating. For example instead of hardcoding multiple `ORACLE_SID` in a DB script, a property `ORACLE_SID` can be set on each of the DB machines. When the script runs against a particular DB machine, every occurrence of `@@ORACLE_SID@@` is expanded to the correct value for that machine.

Creating machine level properties (variables which are available at runtime inside the workflow whenever a workflow is run against the machine)

- To add a property to db-prod1 click on db-prod1 (figure 1.13)

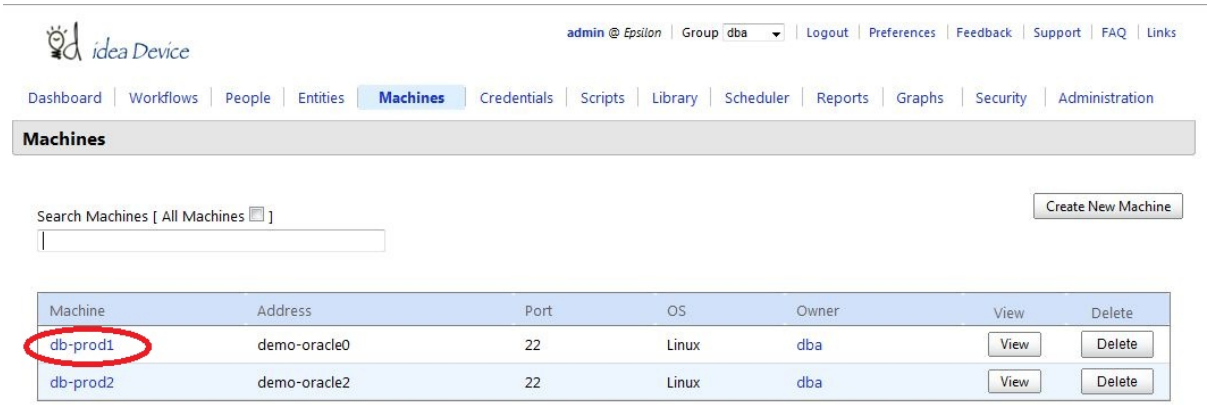
idea Device
admin @ Epsilon | Group dba | Logout | Preferences | Feedback | Support | FAQ | Links
Dashboard | Workflows | People | Entities | Machines | Credentials | Scripts | Library | Scheduler | Reports | Graphs | Security | Administration
Machines
Search Machines [All Machines]
Create New Machine
Machine | Address | Port | OS | Owner | View | Delete
db-prod1 | demo-oracle0 | 22 | Linux | dba | View | Delete
db-prod2 | demo-oracle2 | 22 | Linux | dba | View | Delete

Figure 1.13: Create machine properties

- Click on Add Property (figure 1.14)
- Create a new property called `ORACLE_HOME` with a value `/u01/app/oracle/product/11.2.0/xe`. The variable `ORACLE_HOME` can be accessed in-

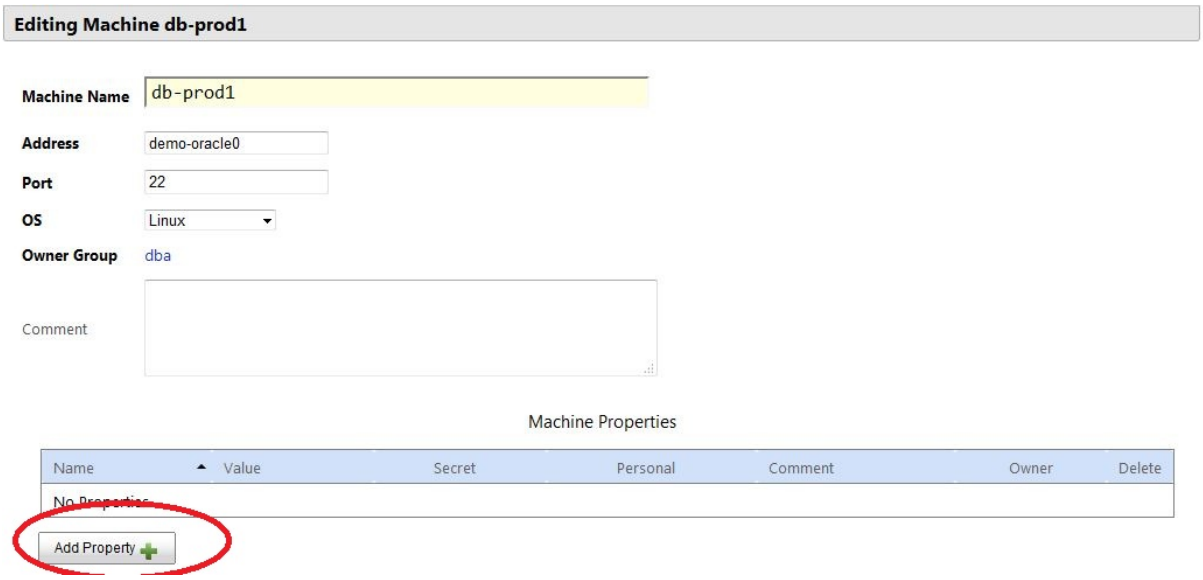

Figure 1.14: Create machine properties

side a workflow as `@@ORACLE_HOME@@`, in the workflows/tasks and the child tasks which have the machine db-prod1 is associated (figure 1.15)

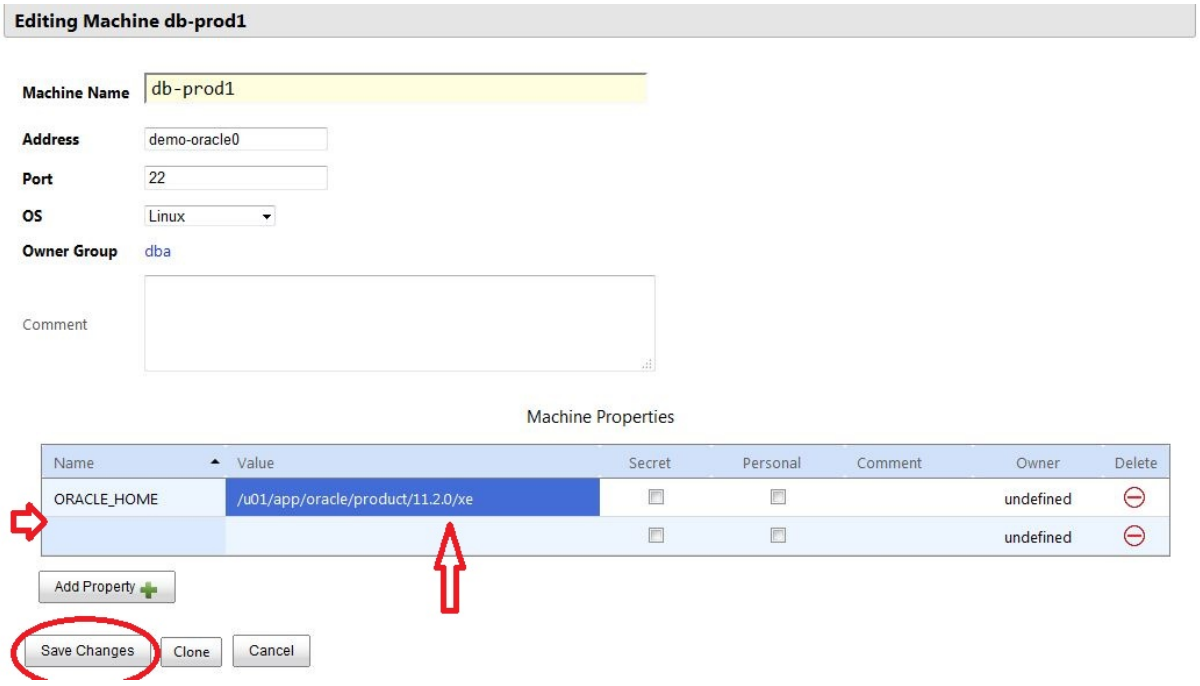

Figure 1.15: Create machine properties

1.8 Create Entities

An abstract object used for grouping purposes. For eg: RAC DB Nodes, All machines with WebSphere installed or All my critical machines. Entity is useful when you want the workflow to operate on a logical or functional collection of machines.

- To create a new entity click on tab- Entities and press the button- Create New Entity (Figure 1.16)

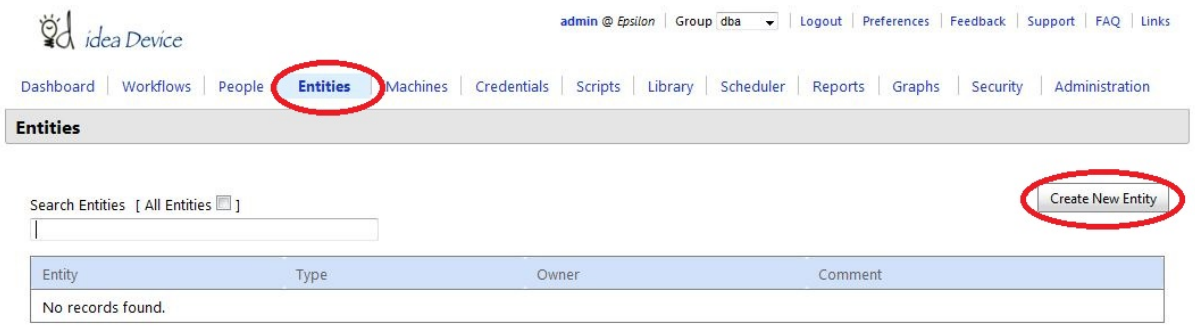

Figure 1.16: Create Entity

- After adding the entity, go back and click the entity name for edit The property tab can be used to create variables which can be used in the workflow. Use Associate Machines button to add machine to this entity. This can also be used to associate credentials to machines (figure 1.17)

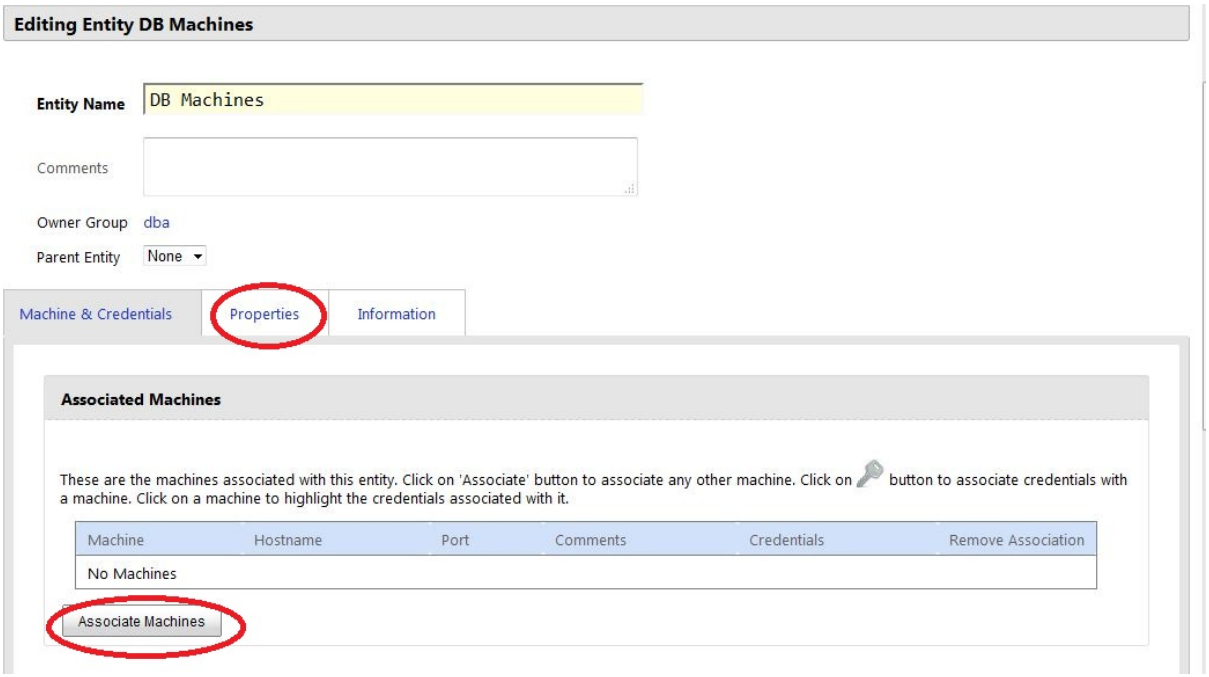

Figure 1.17: Create Entity

- The associate machine dialogue shows the list of machines which this group has access to (figure 1.18)

- After associating the machines, click on the key symbol to associate a credential to the machine. All the credentials are shown, choose the ones to associate the machine with (figure 1.19)

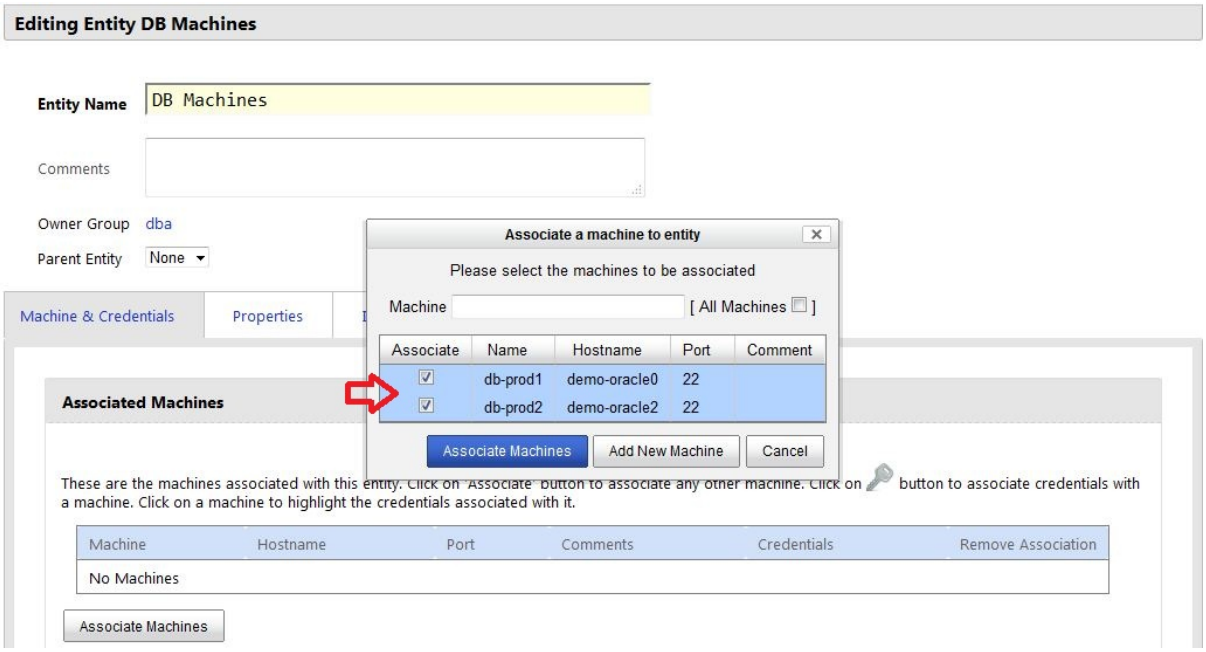

Figure 1.18: Associate Machines

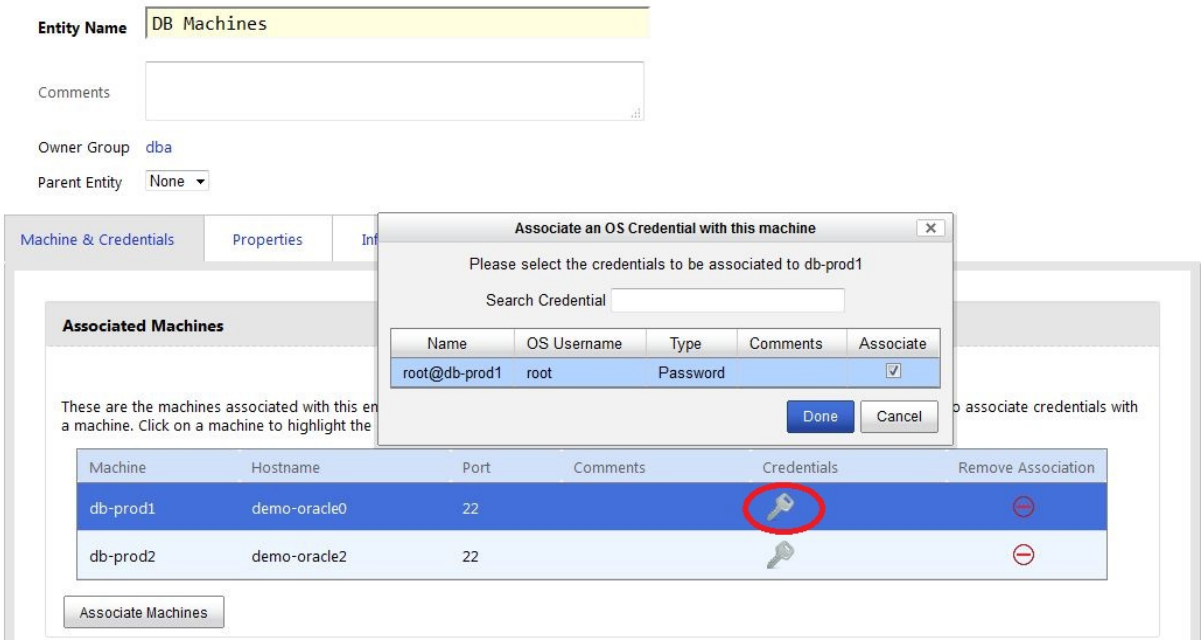

Figure 1.19: Associate Credentials

1.9 Create a Workflow

- To create a work flow, select the workflow tab and click on the button Create a Workflow' (figure 1.20)
- Enter a name for the workflow and press the button Save and Continue (figure 1.21)
- Different tabs are used for setting various characteristics of the workflow.

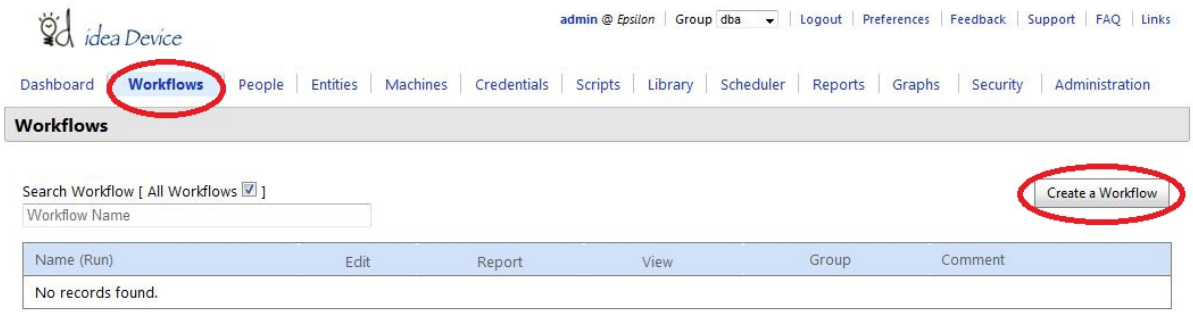

Figure 1.20: Create a New Workflow

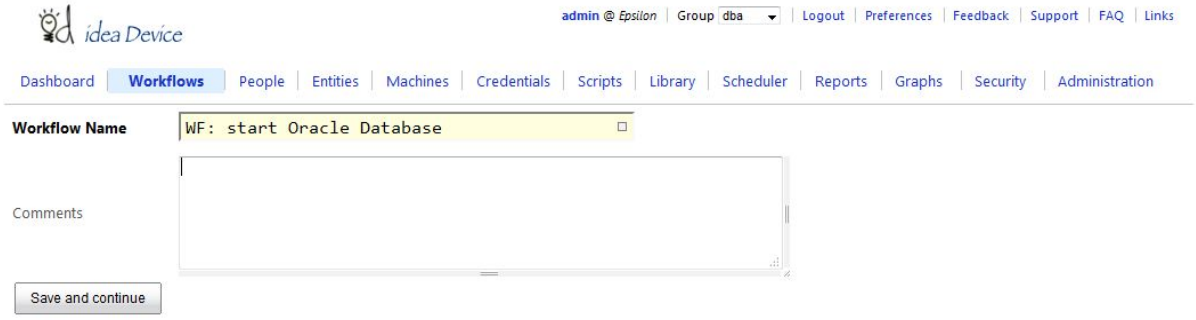

Figure 1.21: Create a New Workflow

  - Tasks tab is used to create/manage tasks.
  - Entity.M.C tab is used to associate entity and machine credentials to this workflow.
  - Security tab is used to grant permissions to other users of this workflow.
  - Workflow Arguments tab is used to take runtime inputs from the users, which are stored in the variables which can be referred in the scripts.
  - Properties tab is used to create variables which can be referred in the scripts
  - Messages tab helps in setting a descriptive success/warning/error messages
  - Exit Status helps in setting the exit status, based on exit code or pattern matching

- Under the task tree tab use the + to create a new task (Figure 1.22)
- The list of values shows different task types, select Run Shell and press Add (Figure 1.23)
- Enter the name of the task. Since we want to use a shell script to do the startup we retain the type as Shell Script. Press Edit to enter the script (Figure 1.24)
- Enter the script to start the database. Here `@@ORACLE_HOME@@` is a property in the machine and `@@start_type@@` is a runtime variable created using workflow arguments and entered by the user at runtime (Figure 1.25)
- The script is also saved in a file, which can be reused in other workflows or tasks (Figure 1.26)

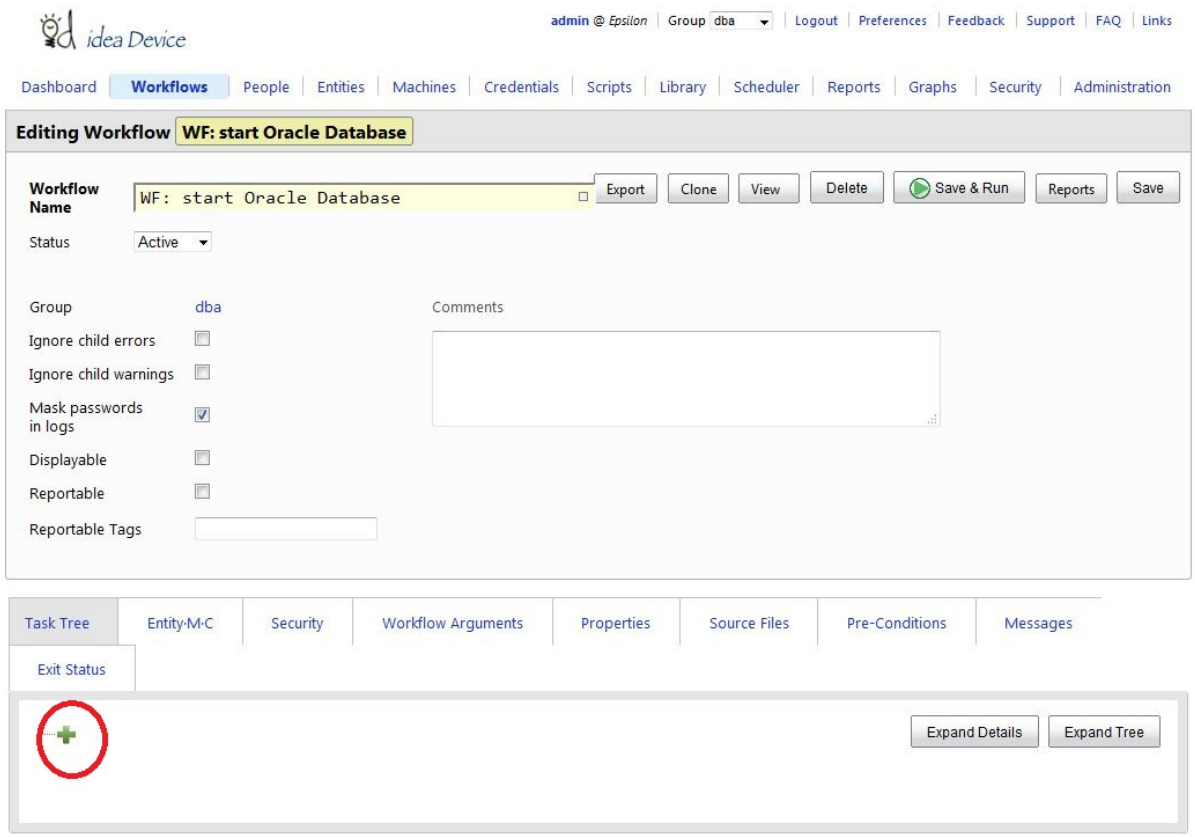

Figure 1.22: Create a New Task

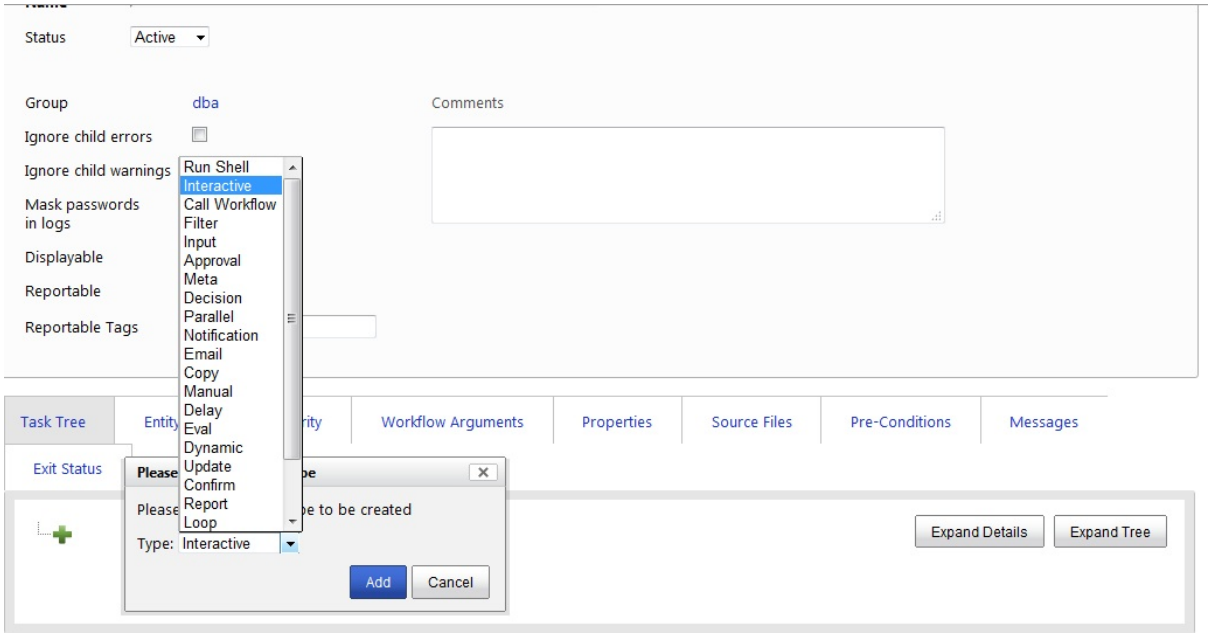

Figure 1.23: Create a New Task

- Click on the Show Advanced button to set the success/failure criteria for the task (Figure 1.27)

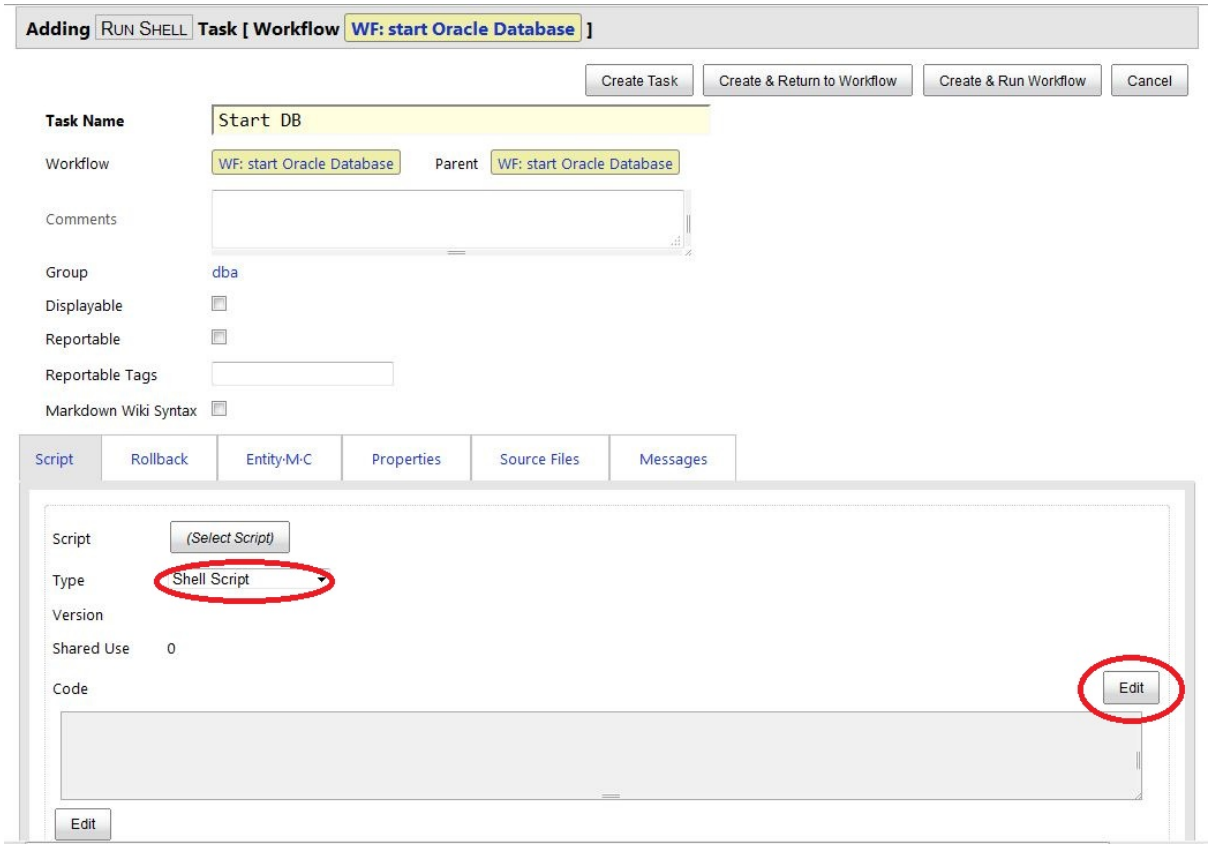

Figure 1.24: Create a New Task

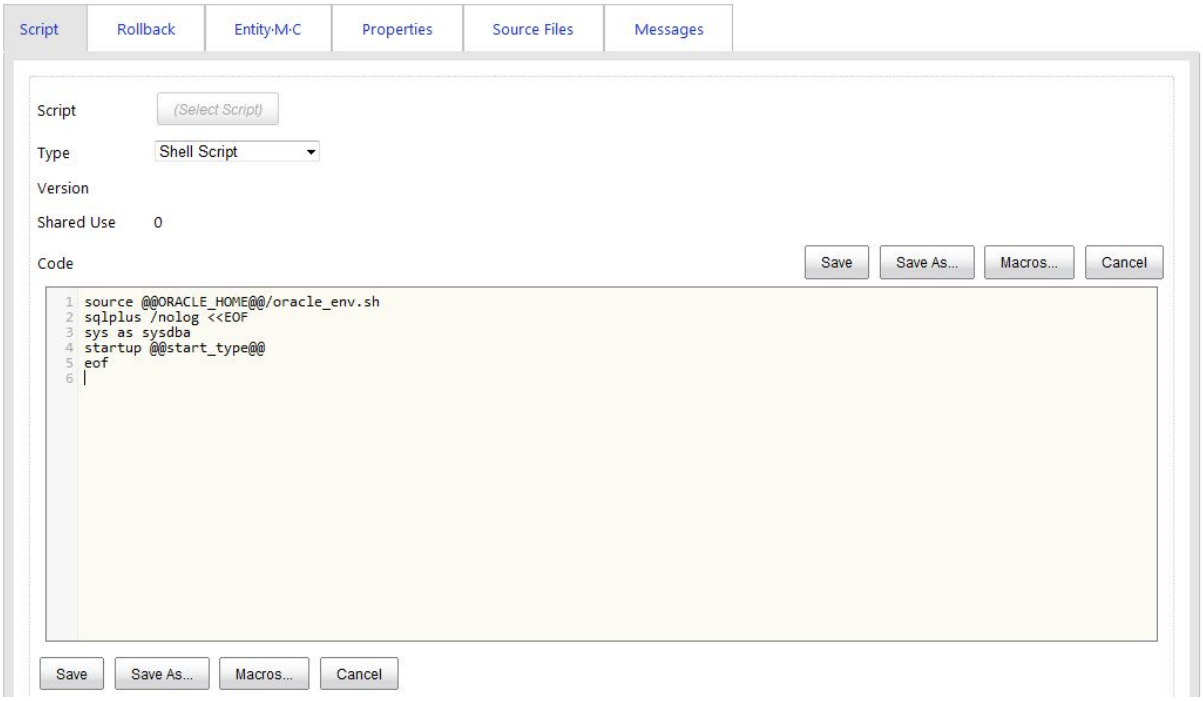

Figure 1.25: Create a New Task

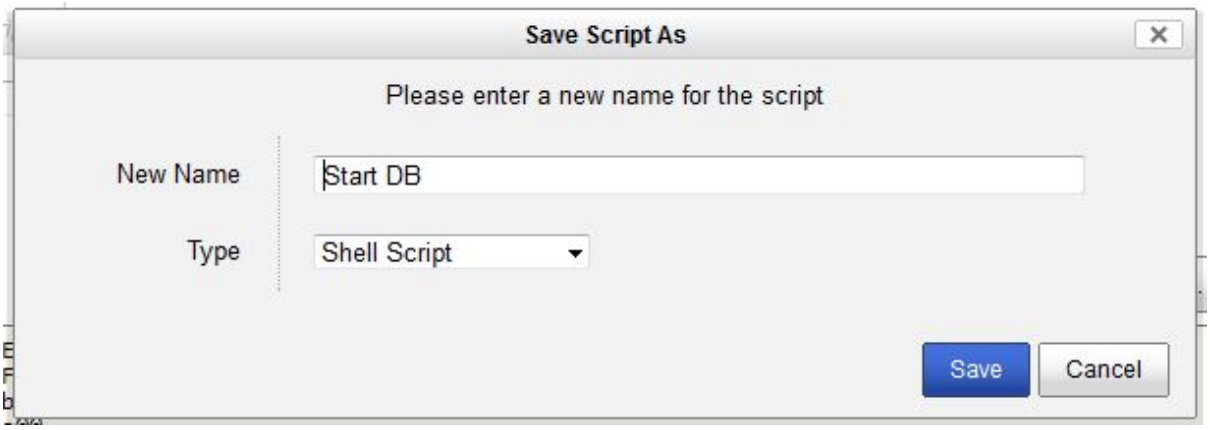

Figure 1.26: Create a New Task

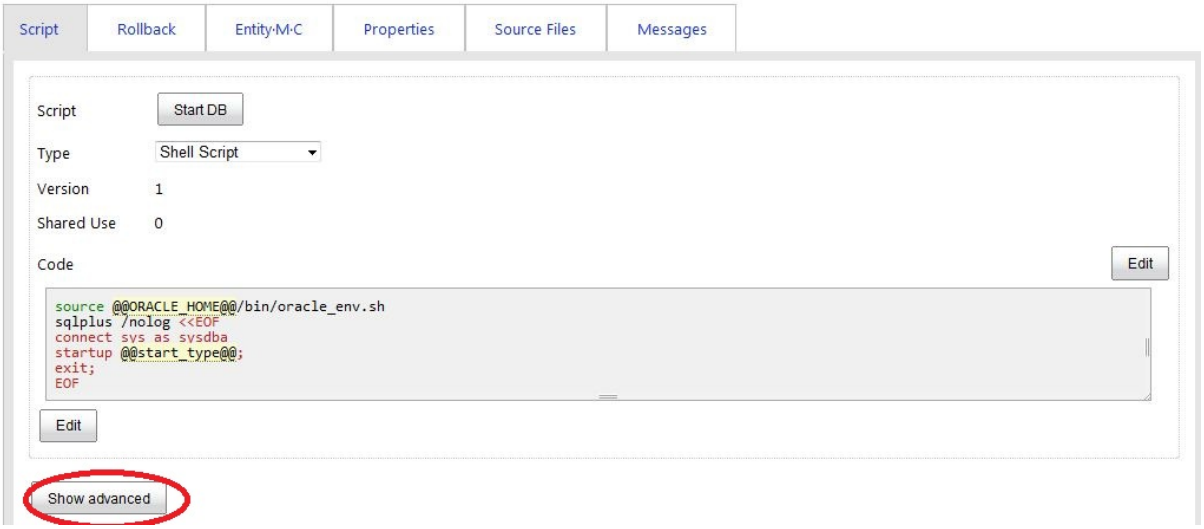

Figure 1.27: Create a New Task

- Enter the success pattern as Oracle instance started, without the quotes. Let all the values of Exit Code lead to success by setting the value to *. Here if the task log does not contain the success pattern, the task is deemed a failure. Save the changes (Figure 1.28)
- Create a workflow argument called `start_type`, which is entered at the run time by the user. Click on save and return to workflow. Click on tab Workflow Arguments (Figure 1.29)
- Click on Add Parameter in the In Parameters and add the following Name: `start_type`, Label: Enter the Database startup type, Default Open, type:list, values: Open, Mount then press save workflow (Figure 1.30)
- Now we are ready to run the workflow.Press save and Run button. You will get the workflow run screen. Enter the Machine and credentials by pressing the change button (Figure 1.31)
- Start the workflow by pressing the Start button The following is the run screen, showing the status of the workflow. The individual task log can be accessed by double clicking on Start DB (Figure 1.32)

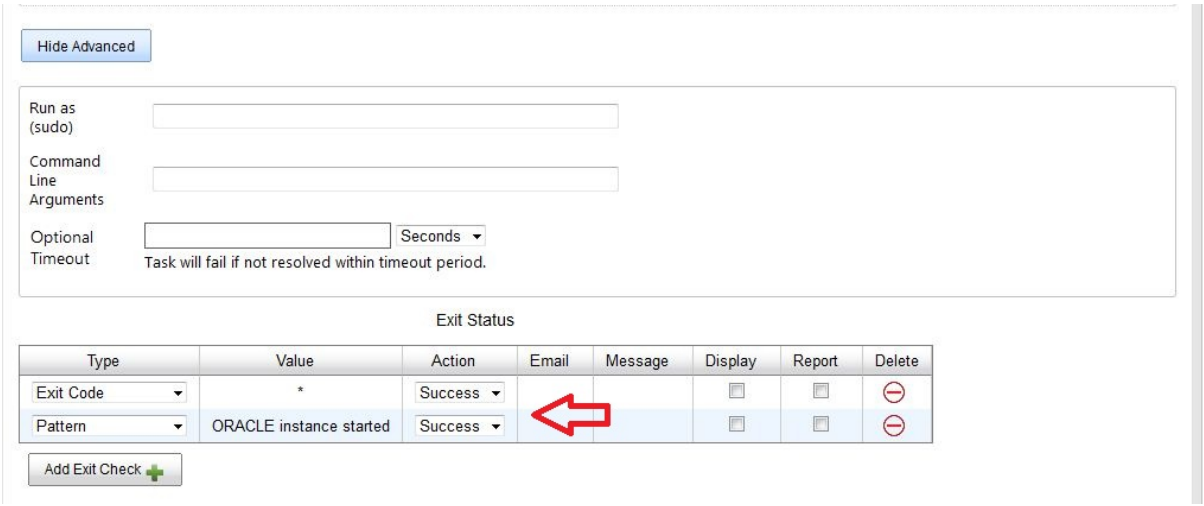

Figure 1.28: Create a New Task

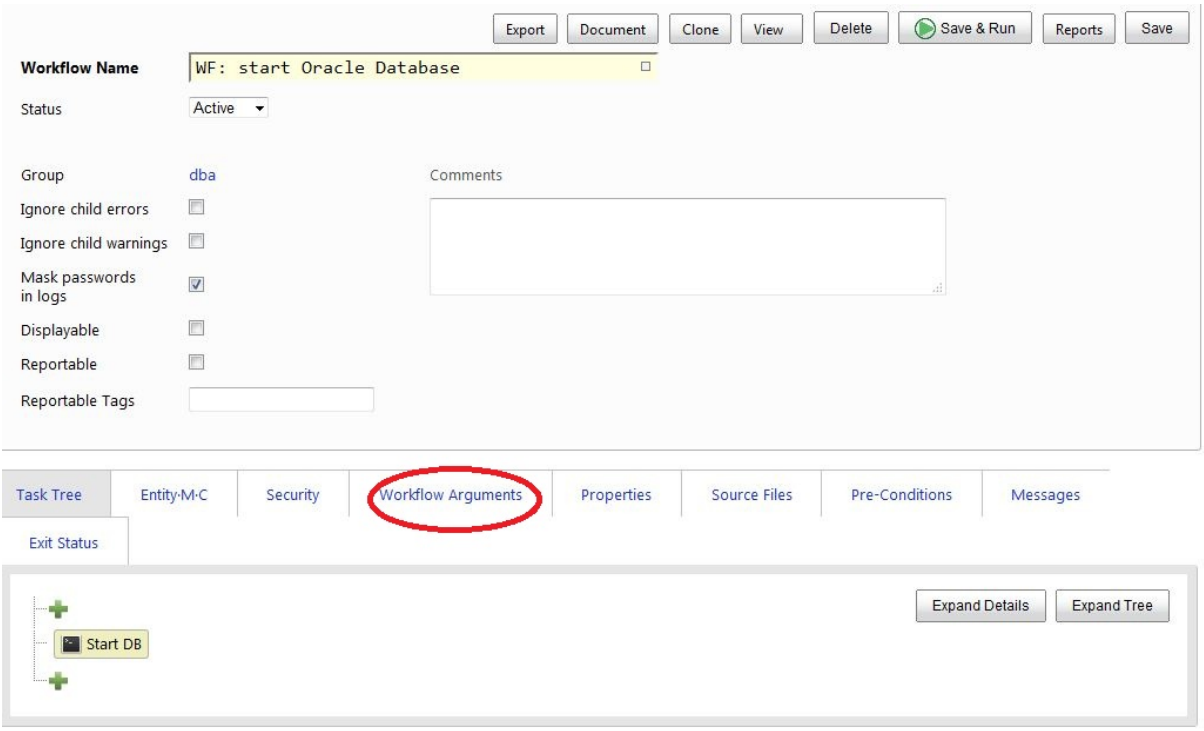

Figure 1.29: Create a Workflow Argument

- The Task Information shows the output log and the pattern matched, The script which was run, including the values substituted for variables, `ORACLE_HOME` and `start_type`, Properties, which is the list of variables used, and the task run information (Figure 1.33)

- The Reports tab shows all the workflows that are run and viewable by this account

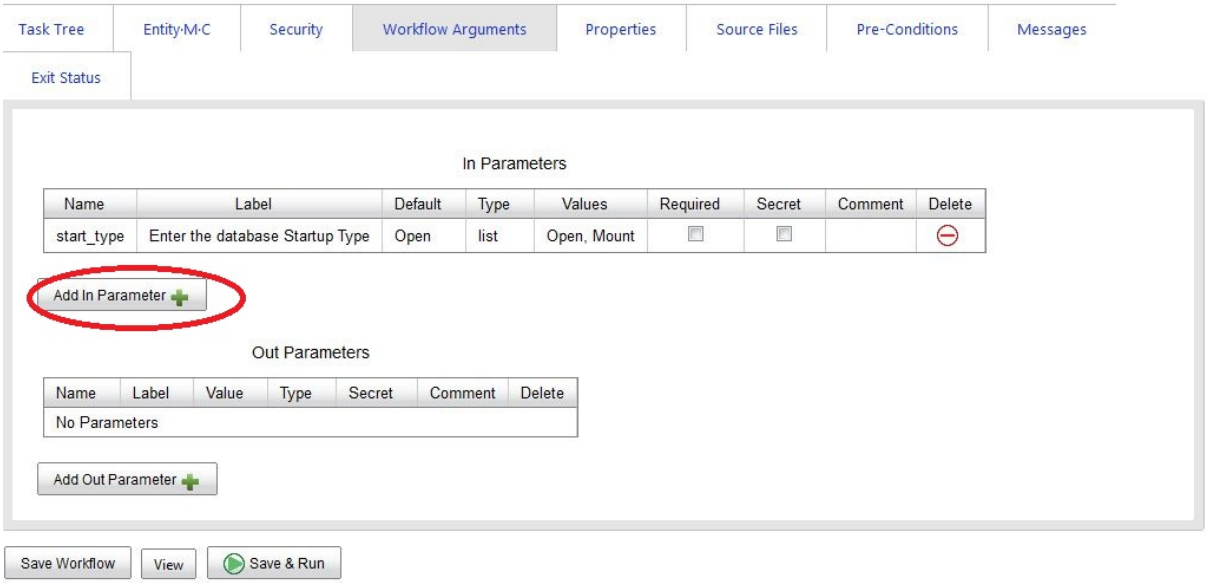

Figure 1.30: Create a Workflow Parameter

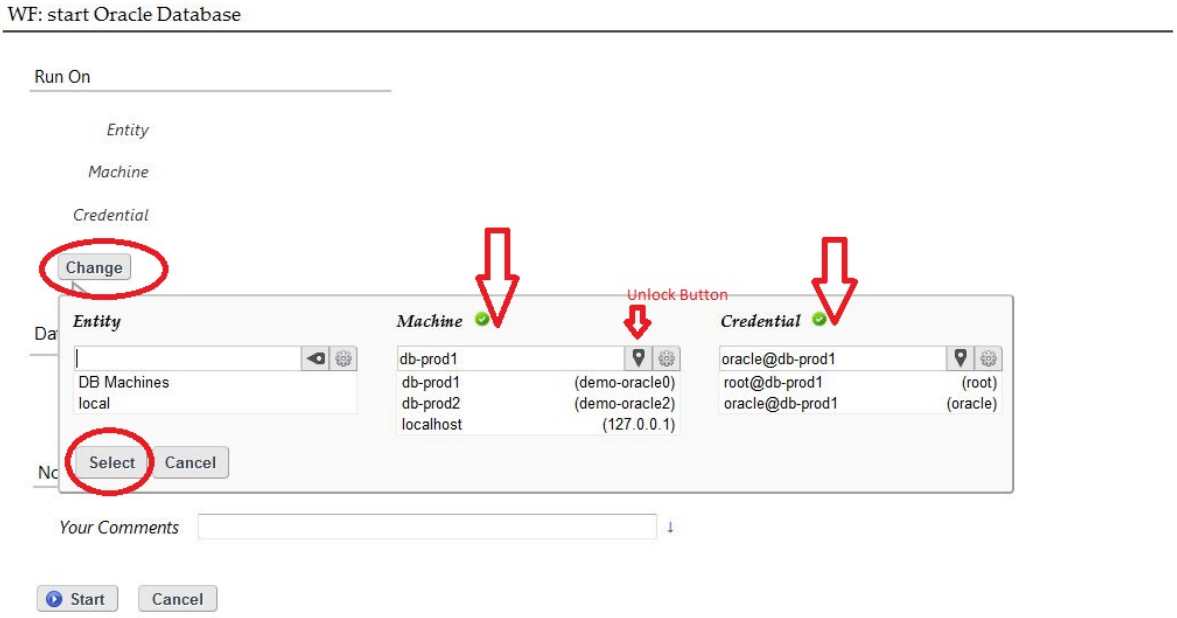

Figure 1.31: Run a Workflow

1.10 Using EMC Macros, Input task, Loop task, call workflow task

We now modify the above workflow which is used to start a database, to start multiple databases. We would like to start all the databases in the machine grouped under a particular entity. The following illustration explains how to achieve this.

- Make sure that an entity called DB Machines is created and has all the required machines and the credentials associated (Figure 1.34)

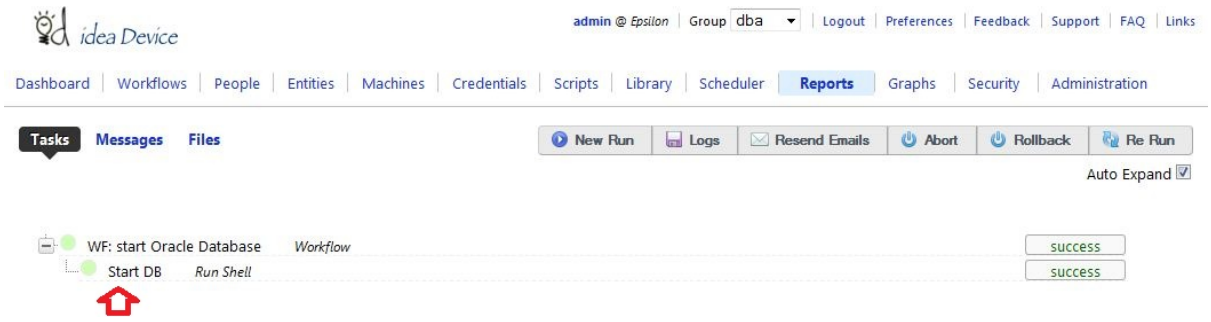

Figure 1.32: Run a Workflow

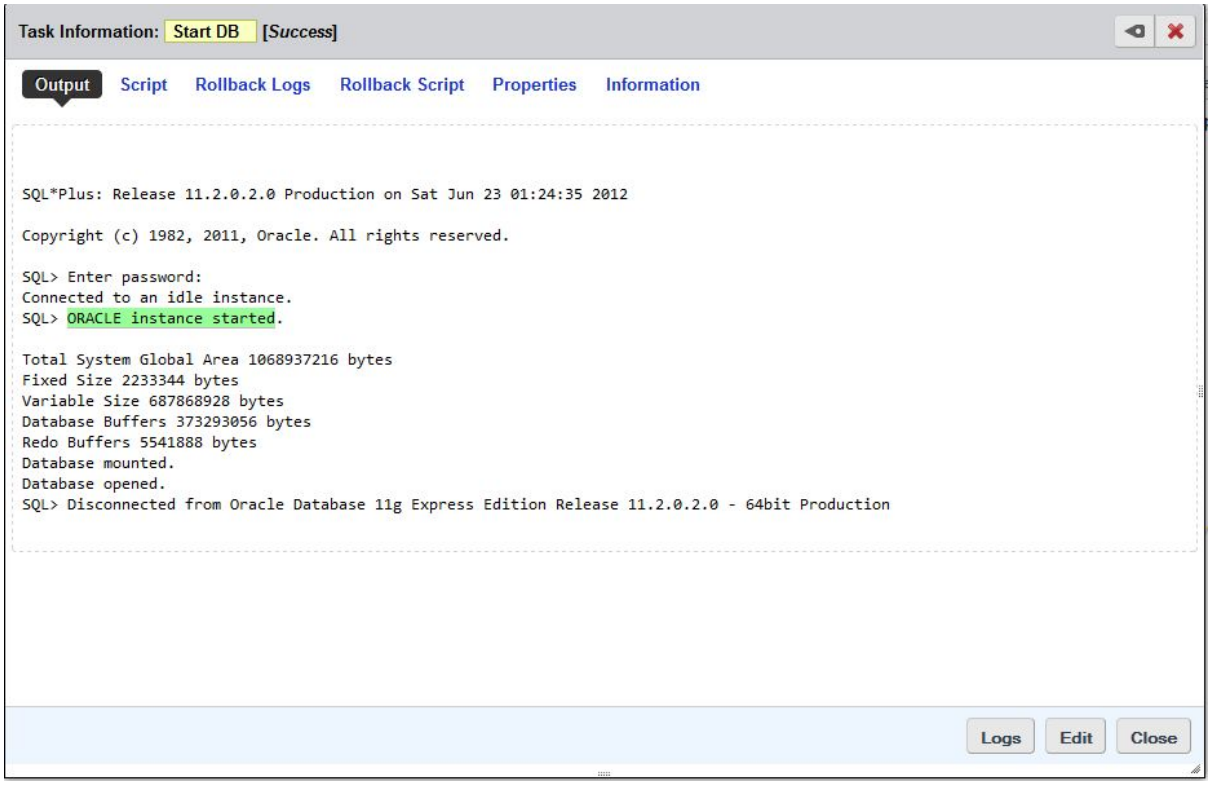

Figure 1.33: View Run Log

- We need to create the following tasks to perform the actions
  1. An input task to take input `start_type`, which is the database startup type that is desired.
  2. A loop task to fetch the machines associated with an entity and perform certain child tasks
  3. A Meta task acts as a parent task to group together certain children tasks, which need to be executed in the same Entity, machine, credential context. This task uses what is called EMC Macros to set the current Machine/Credential context.
  4. A call workflow task to call a workflow called WF:start Oracle database, which can start one single oracle database (Figure 1.35)
- An Input Task allows you to ask the user running the workflow for input values or selections

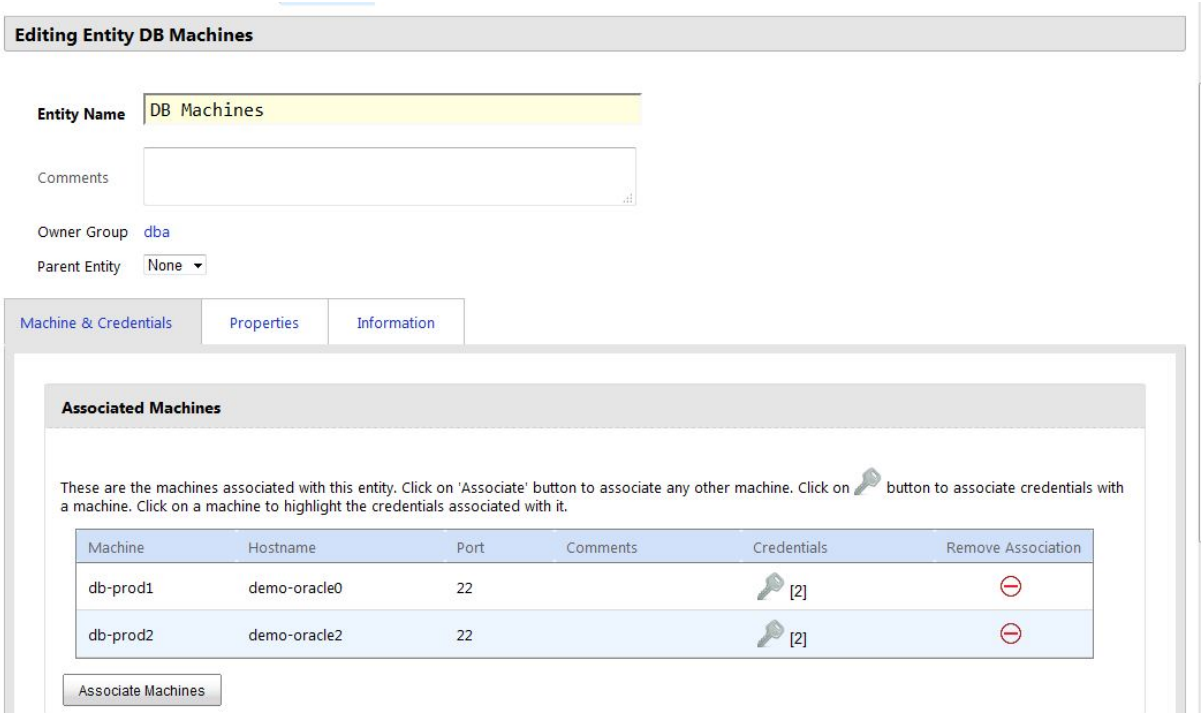

Figure 1.34: Add to Entity

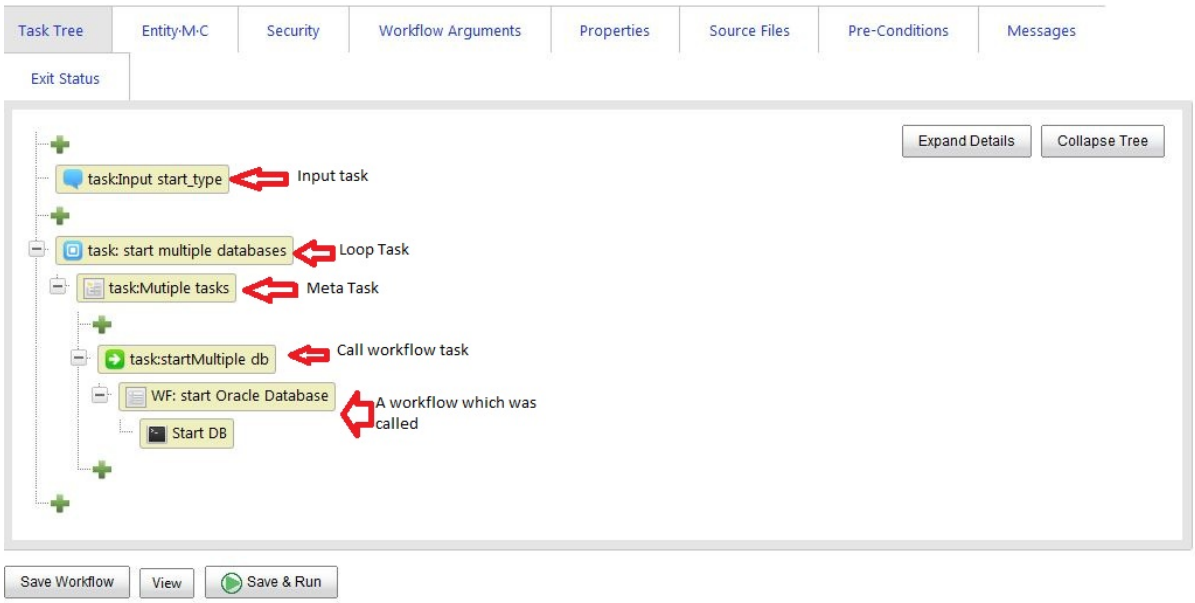

Figure 1.35: Create Tasks

through the browser UI. Input tasks can render text boxes with defaults, checkboxes, file uploads, single selections and multiple selections. Input task stores the values into properties named by task output. Input task can run shell scripts or use Static text to feed custom UI into Input task output. More details about the input task type can be found in the faq section of Epsilon product portal (Figure 1.36)

- Create a task of type loop. Select For loop as the loop type. For the loop over list we want

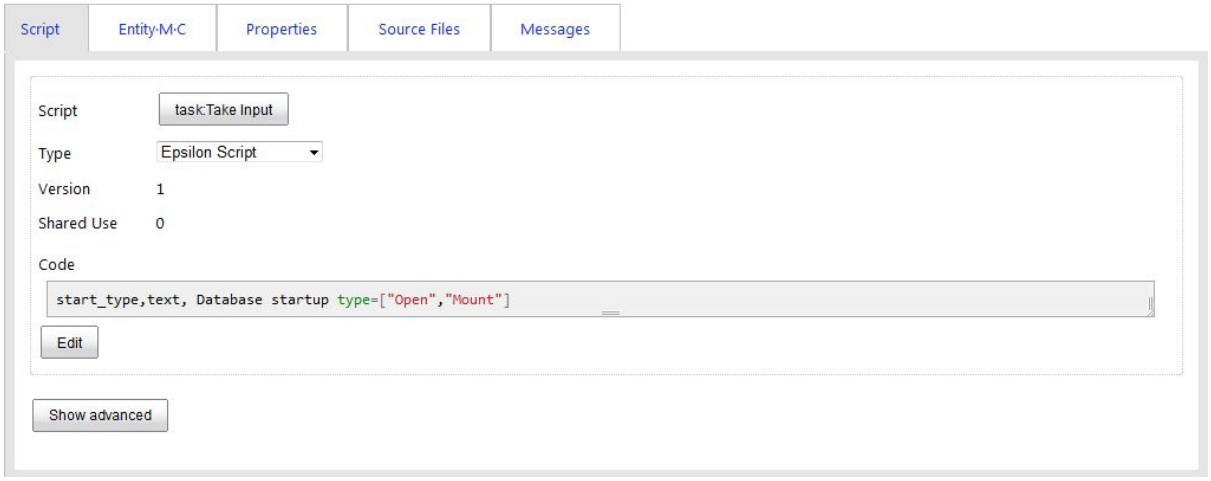

Figure 1.36: Input Task

to fetch all the machines associated to one entity, which is given at the runtime. This can be done using an inbuilt macro called `AZ_LIST_Entity`. We use THIS to indicate the current entity which has been set by the user. Hence `@@AZ_LIST_Entity(name="THIS").get(Machine(name))@@` gives the list of machines delimited by ,. Loop variable `mch_name` holds one value of the machine for each iteration of the for loop. Speedup factor 1 indicates that each iteration is to be executed sequentially. A value greater than 1 would execute the iterations parallel. For example if Speedup factor=2 , two machines at a time will be simultaneously picked up for db startup (Figure 2.9)

- Lastly we need to create a meta task which is used to set the machine and credential context for the children tasks. Create a meta task called task:Mutiple tasks. Click on the Entity.M.C tab and click the icon next to the Machine or Credential (Figure 1.38)

- Click on the macro button indicated (Figure 1.39)

- Enter the EMC macros, as documented in the faq. The loop variable `@@mch_name@@` contains the machine name that needs to be set as the current context. Enter `@@mch_name@@` under the Machine macro. Next the credential associated with this machine needs to be fetched. That is done using the macro `AZ_LIST_Machine`. `Name~mch_name` would expand the variable `mch_name` and fetch the properties of a machine with `name=<contents of mch_name>`. Finally we would like to get the credential with a username=Oracle, as this machine may also have other credentials such as root associated with it, but we would like to use the username=oracle for database related tasks. The final form of the macro which is given as Credential macro is as follows `@@AZ_LIST_Machine(name~"mch_name").get(Credential(name), uid="oracle"))@@` (Figure 1.40)

- Finally add a task type called call workflow as a child of the meta task. Choose the workflow called WF: start oracle database, by clicking on the list of values (Figure 1.41)

- Save the workflow and now you are ready to run the workflow

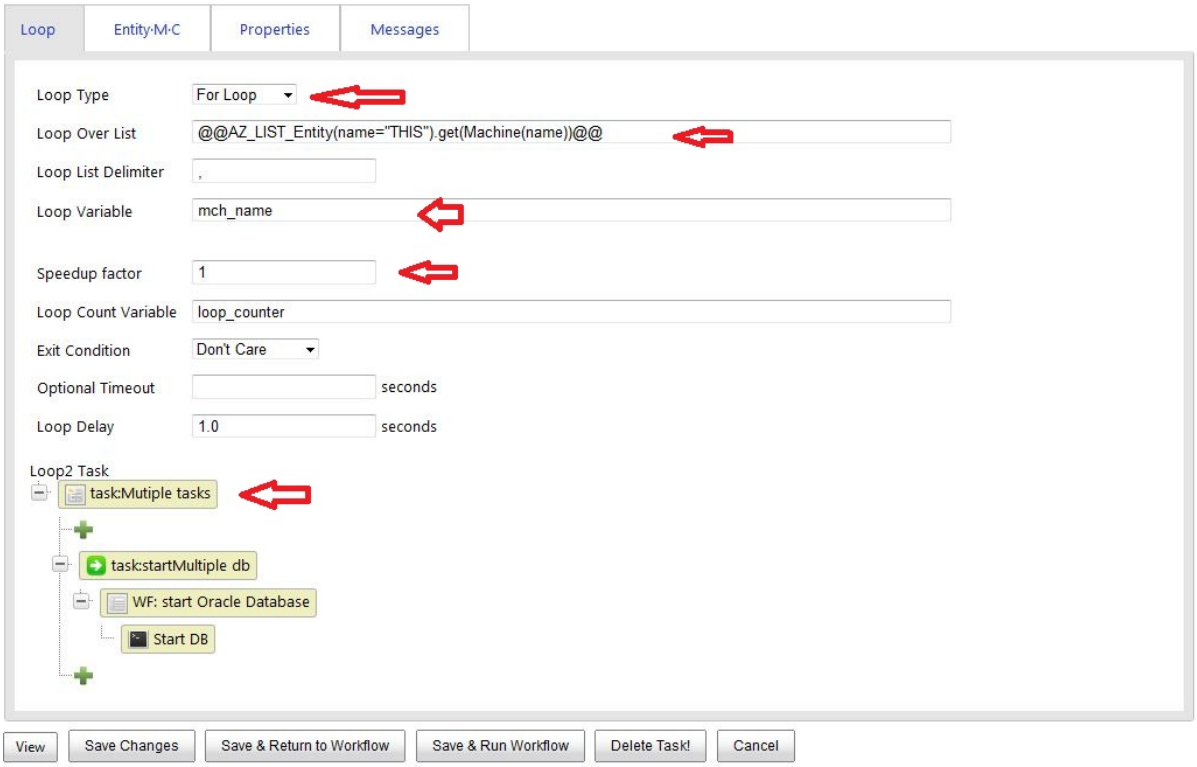

Figure 1.37: Loop Task

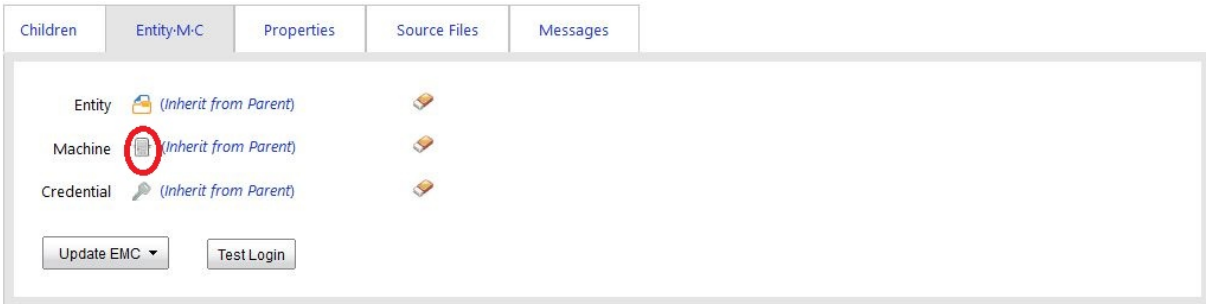

Figure 1.38: EMC Macro

1.11 Scheduler

Epsilon comes with an advance scheduler to run any workflow repeatedly at a predetermined time. Figure 1.42 shows the screen to create a workflow schedule. Workflow can be scheduled to run at specific date and time, repeatedly run on one or multiple specific hour,minute, day, month, year, and weekday. Scheduler can also be scheduled to run till a particular date and time.

The macro variable AZ_SCHEDULED is set to 1 when a workflow is executed via scheduler. This can be used to execute a specific logic for scheduled workflow.

Any cron type of expressions are allowed while specifying the schedules. In figure 1.43, pick the advanced radio button to enter a cron expression.

Table 1.1 lists some of the cron expressions

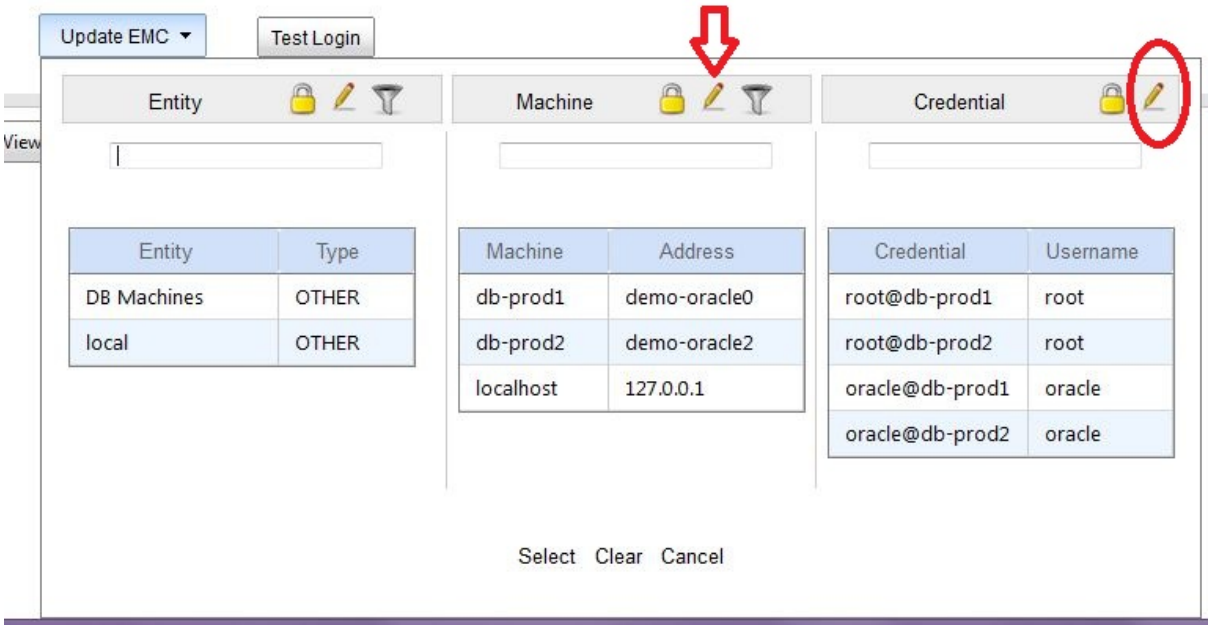

Figure 1.39: EMC Macro

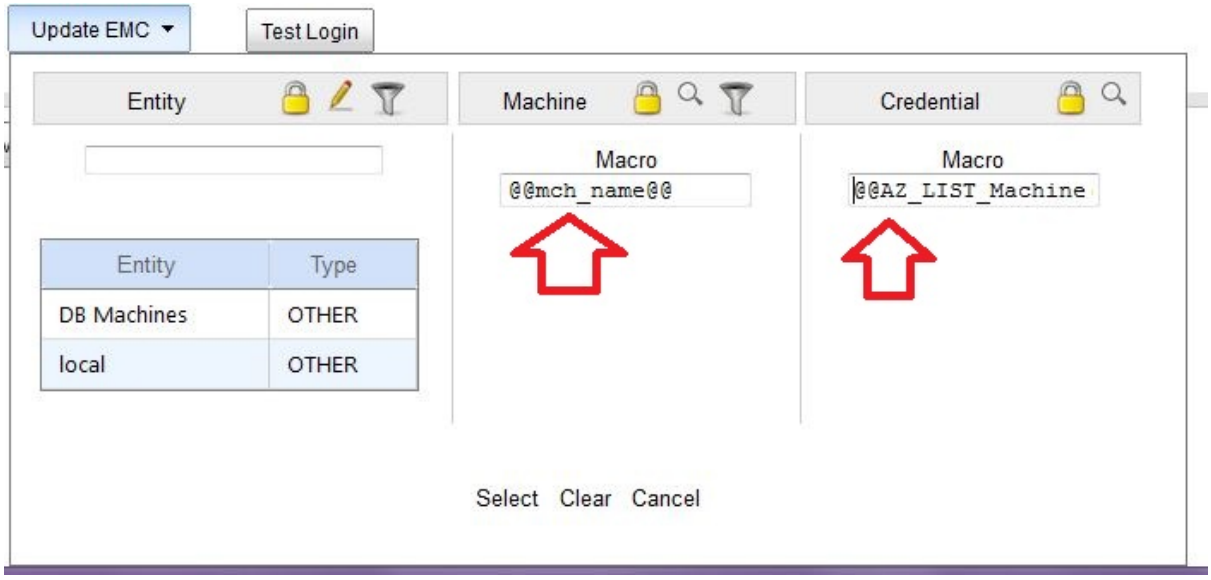

Figure 1.40: EMC Macro

1.12 View the run logs

The workflow run log of any workflow run can be viewed using the "Reports" tab. "Show graph" button in the reports page (figure 1.44) provides a summary statistics, in graphical form, of the workflow runs. Reports tab also has an elaborate query form to search for the past runs of the workflow. The workflow runs can be queried on workflow names, execution dates, workflow run status, groups and users, entity, machine and credentials.

The run log details can also be exported into excel or csv formats.

To view the run log of a particular workflow run, click on the corresponding "Success" or "Failure labels under the "Status" column field (view figure 1.45). This would display the whole

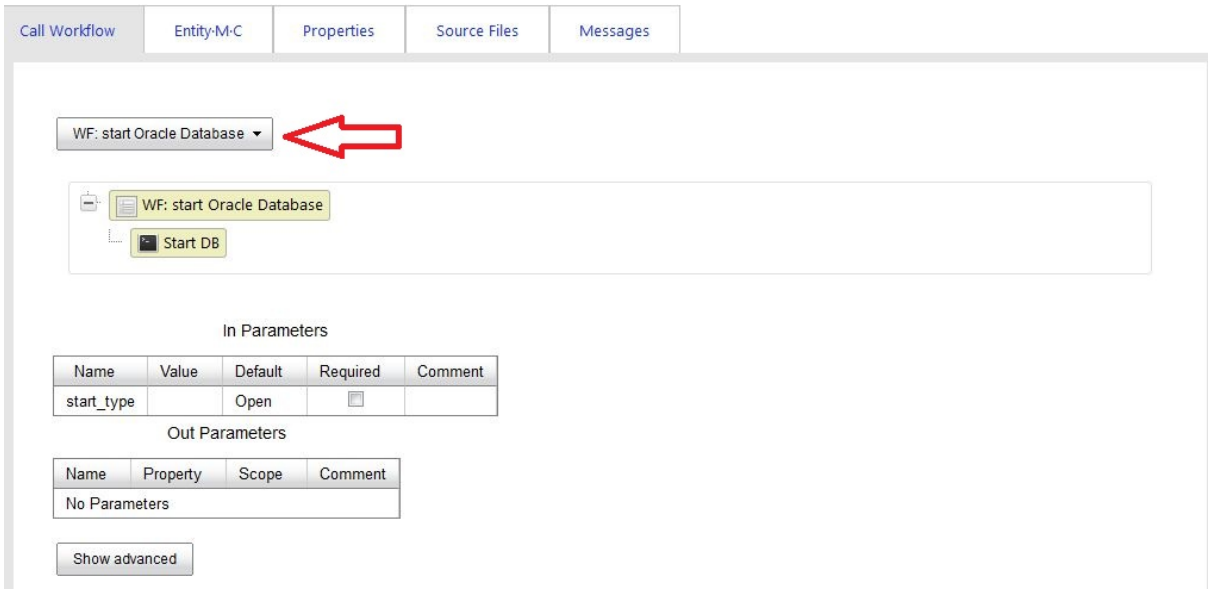

Figure 1.41: Calling another workflow

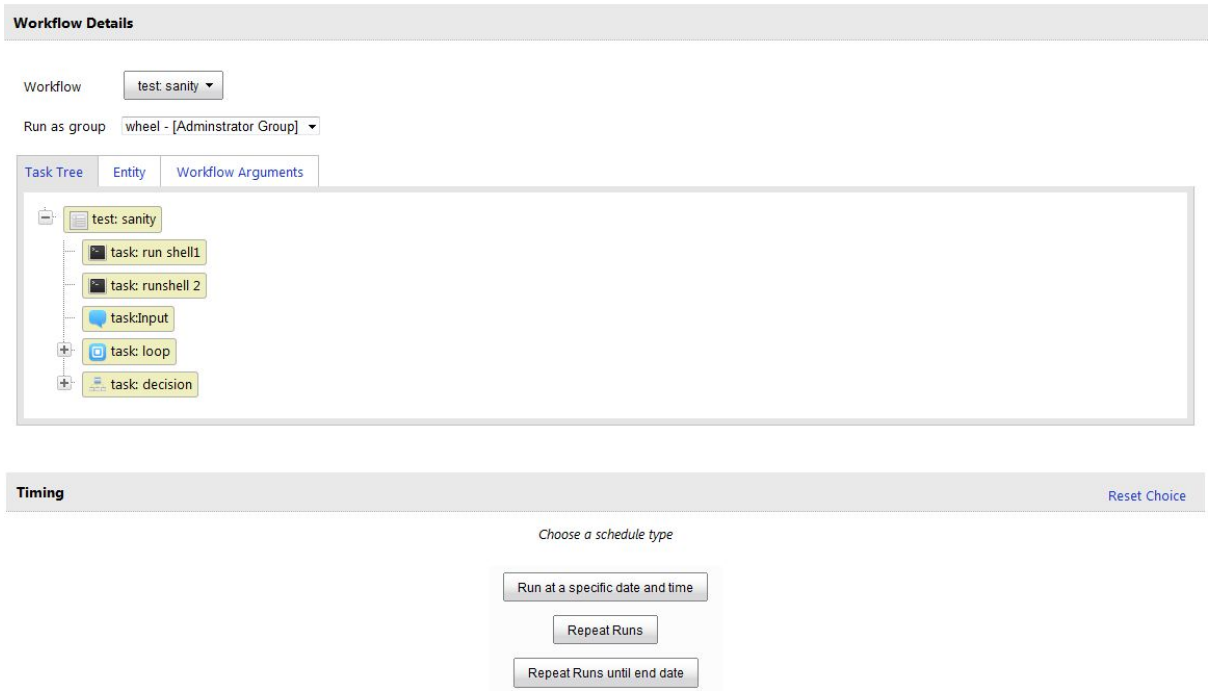

Figure 1.42: Create a Schedule

task tree and the status of each task at a glance in a nice tree format, as shown in figure 1.46.

Clicking on the task name would popup a task information page depicted in figure 1.47. The output tab in the task information popup displays the output of the commands executed.

The script tab displays the command executed in the task (figure 1.48), and also the expansion of any macro variable used in the task.

If a rollback script was used to rollback the task, the script is shown in the rollback script tab and the log is shown in the rollback logs tab.

The runtime task properties set is available in the Properties tab (figure 1.49), along with

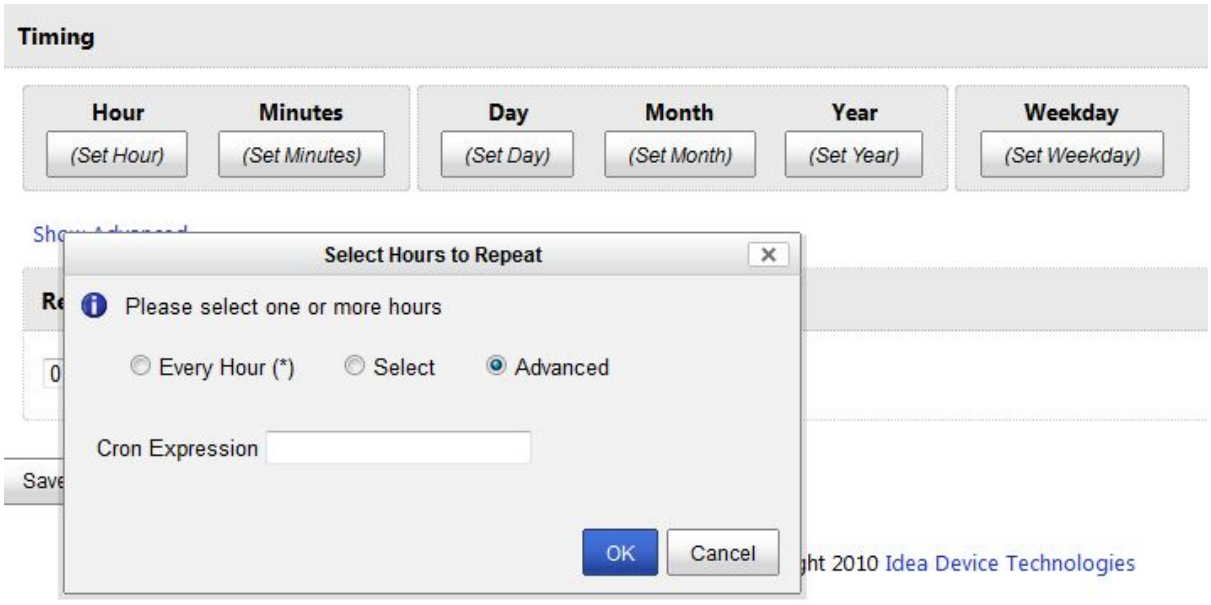

Figure 1.43: Enter a cron style expression

Table 1.1: Example cron expressions

	Minutes	Hours	Day of month	Month	Weekday	Year
Every 30 minutes, during the working hours 9AM to 5PM	*/30	9-17	*	*	?	*
Every two Hours	*	*/2	*	*	?	*
Weekdays at 6:00PM	0	18	?	*	1-5	*
Every last day of month at 9:00 AM	0	9	L	*	?	*

the details about how the property value was inherited.

Finally the Information tab provides details about who executed the task (figure 1.50), when and on which machine, the credential used, and time taken to complete the task, there by providing a complete audit log of the task executed on a machine.

The task log can be exported by pressing the Logs button at the bottom of the task information popup.

1.13 Abort, Rollback, Re Run a workflow

The report tab also provides a mechanism to abort a workflow. Aborted workflows and the workflows which have failed can be restarted by pressing the re Run button. Workflows can be rerun for all the failed tasks or from the last failed task. The rerun can also be done with new inputs. That means if workflow fails due to insufficient diskspace, it is not necessary to rerun the tasks which have been executed successfully. Instead after fixing the problem which caused the task to fail, the workflow can be restarted from that point onwards.

In the workflow edit page, for every task, user can define a rollback script which needs to run, to undo the changes done by the main task script. This rollback script can be optionally

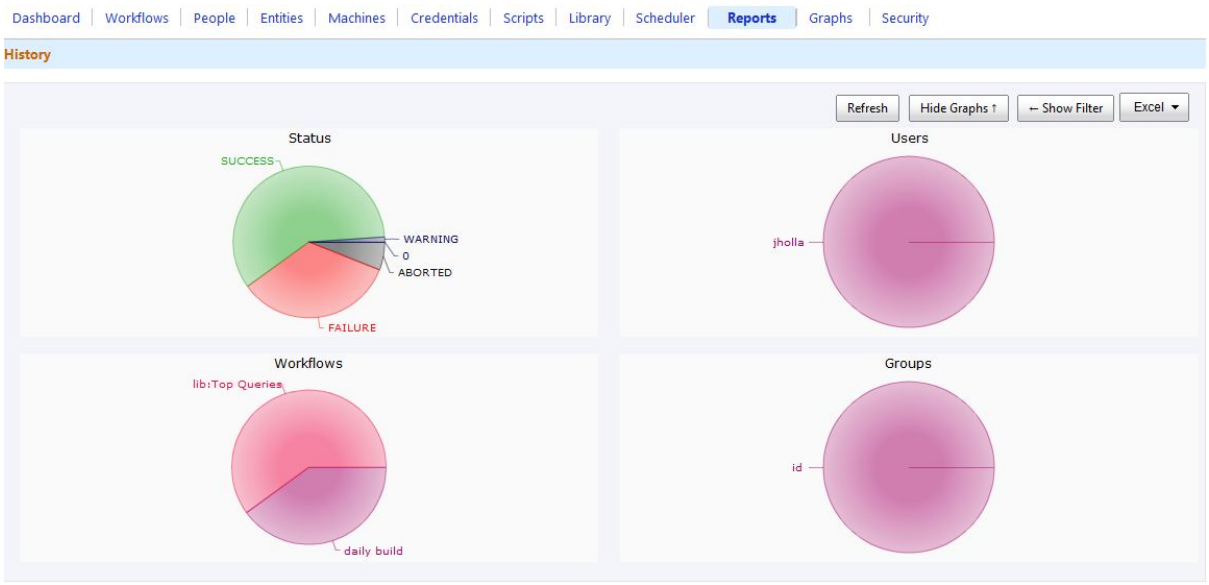

Figure 1.44: Graphical view of the report run status

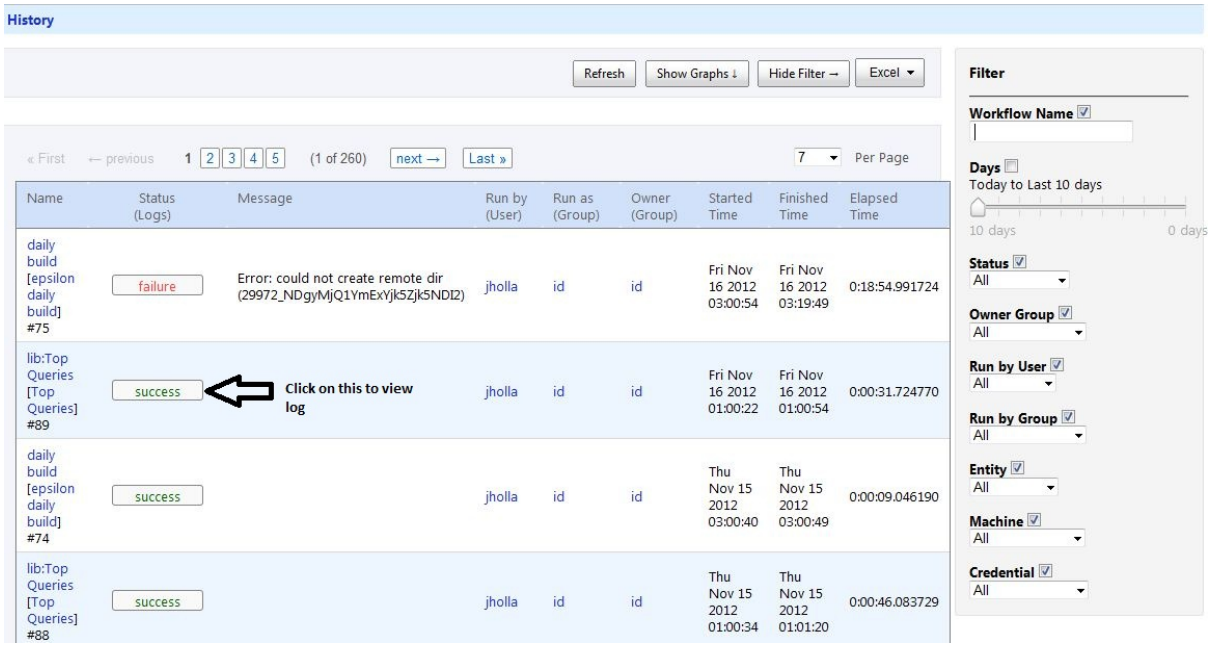

Figure 1.45: Filtering workflow run log

run for the failed tasks, by pressing the "Rollback" button in the reports tab.

Emails communications are sent to the approvers of an approval task and of the executors of manual tasks. The resend emails button can resend the emails to these recipients (figure 1.52).

1.14 Workflow Edit Page

Workflow edit page has many utilities and mechanisms to setup restrictions on a workflow. Some of them are listed below.

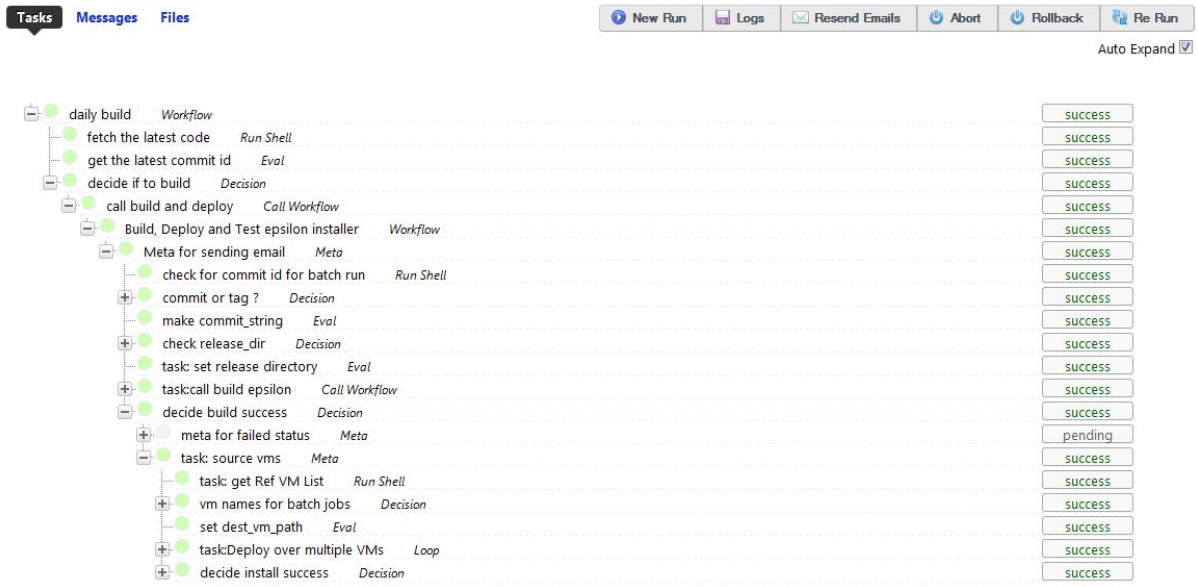
Tasks
Messages
Files
New Run
Logs
Resend Emails
Abort
Rollback
Re Run
Auto Expand
daily build Workflow success
fetch the latest code Run Shell success
get the latest commit id Eval success
decide if to build Decision success
call build and deploy Call Workflow success
Build, Deploy and Test epsilon installer Workflow success
Meta for sending email Meta success
check for commit id for batch run Run Shell success
commit or tag ? Decision success
make commit_string Eval success
check release_dir Decision success
task: set release directory Eval success
task:call build epsilon Call Workflow success
decide build success Decision success
meta for failed status Meta pending
task: source vms Meta success
task: get Ref VM List Run Shell success
vm names for batch jobs Decision success
set dest_vm_path Eval success
task:Deploy over multiple VMs Loop success
decide install success Decision success

Figure 1.46: View workflow run log

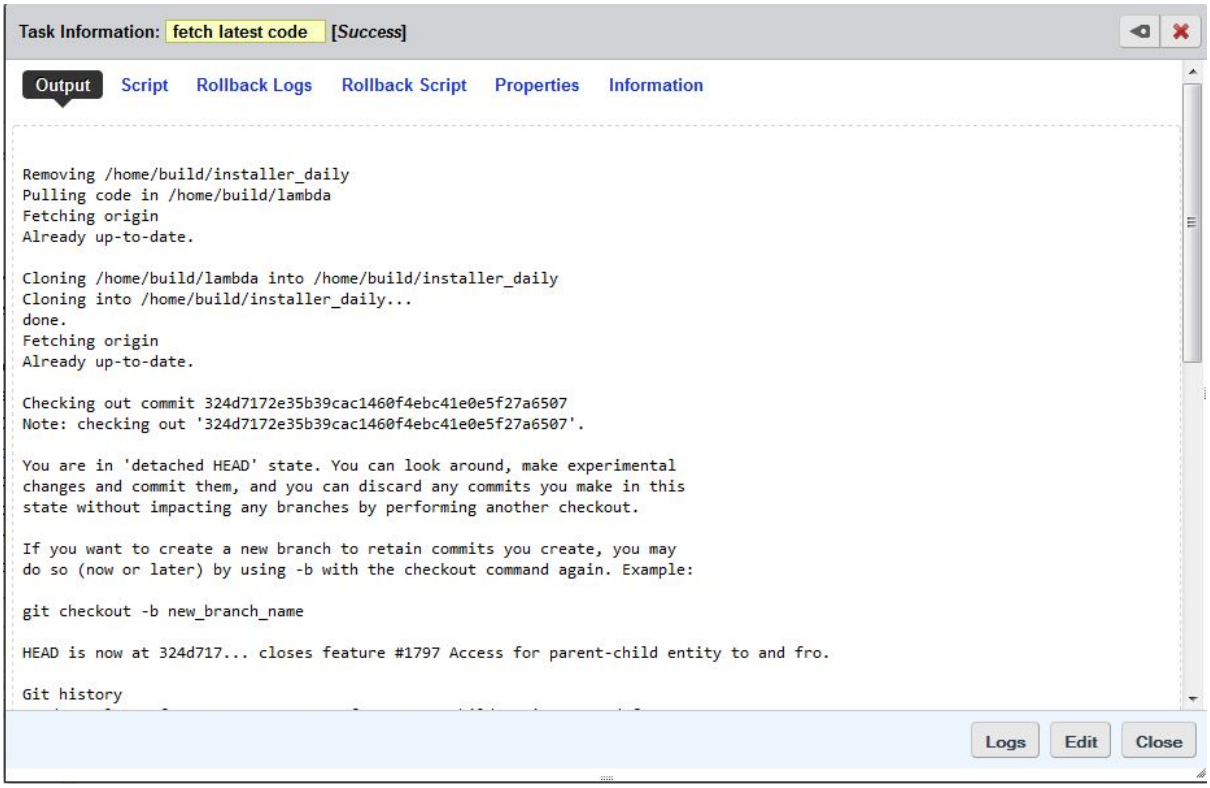
Task Information: fetch latest code [Success]
Output
Script
Rollback Logs
Rollback Script
Properties
Information
Removing /home/build/installer_daily
Pulling code in /home/build/lambda
Fetching origin
Already up-to-date.
Cloning /home/build/lambda into /home/build/installer_daily
Cloning into /home/build/installer_daily...
done.
Fetching origin
Already up-to-date.
Checking out commit 324d7172e35b39cac1460f4ebc41e0e5f27a6507
Note: checking out '324d7172e35b39cac1460f4ebc41e0e5f27a6507'.
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
git checkout -b new_branch_name
HEAD is now at 324d717... closes feature #1797 Access for parent-child entity to and fro.
Git history
Logs
Edit
Close

Figure 1.47: View task output

1.14.1 clone a workflow

A Workflow can be cloned and a new copy can be created by using the clone button in the edit workflow page. Cloning creates a copy of all the tasks and the scripts associated with the tasks In the clone workflow page (figure 1.53) enter the new workflow name and group. Give optional

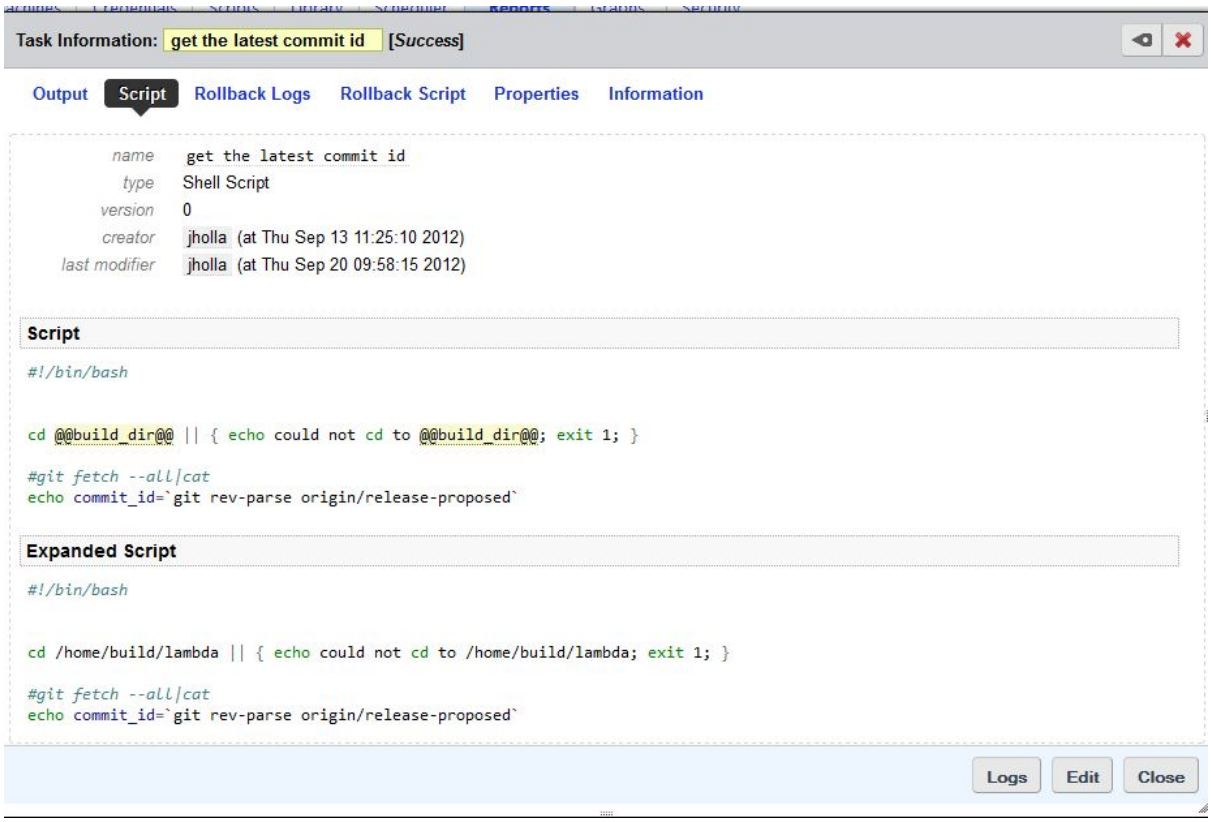

Figure 1.48: View task script

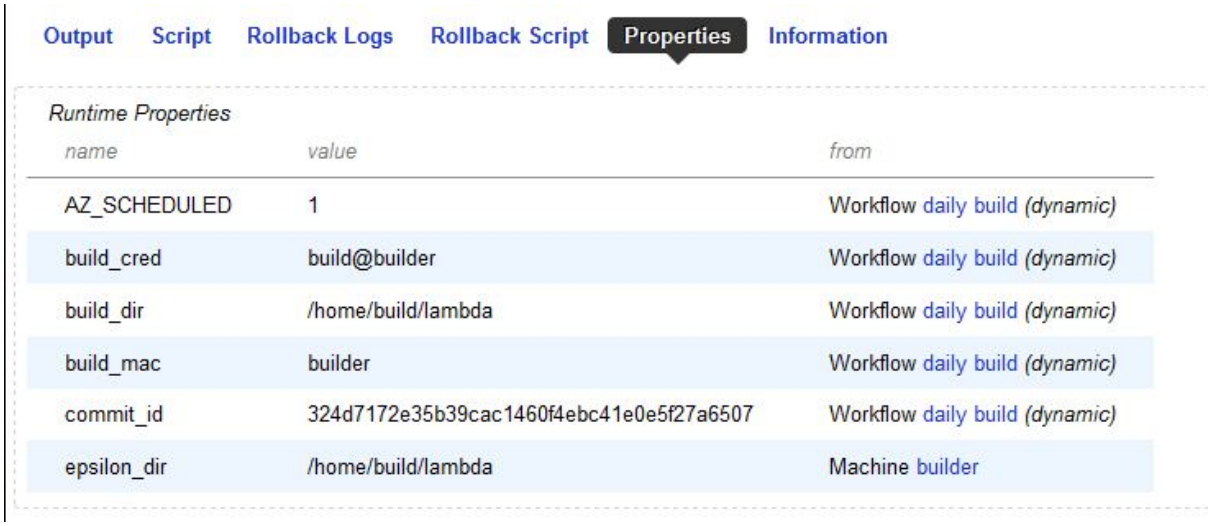

Figure 1.49: View task properties

comments and press "Clone Workflow".

1.14.2 Version Control

Epsilon comes with a version control system to commit a workflow and create a version. A new version of the workflow can be created by pressing the "Commit Version" button in the workflow edit page. Figure 1.54 shows the dialogue when the commit version button is pressed.

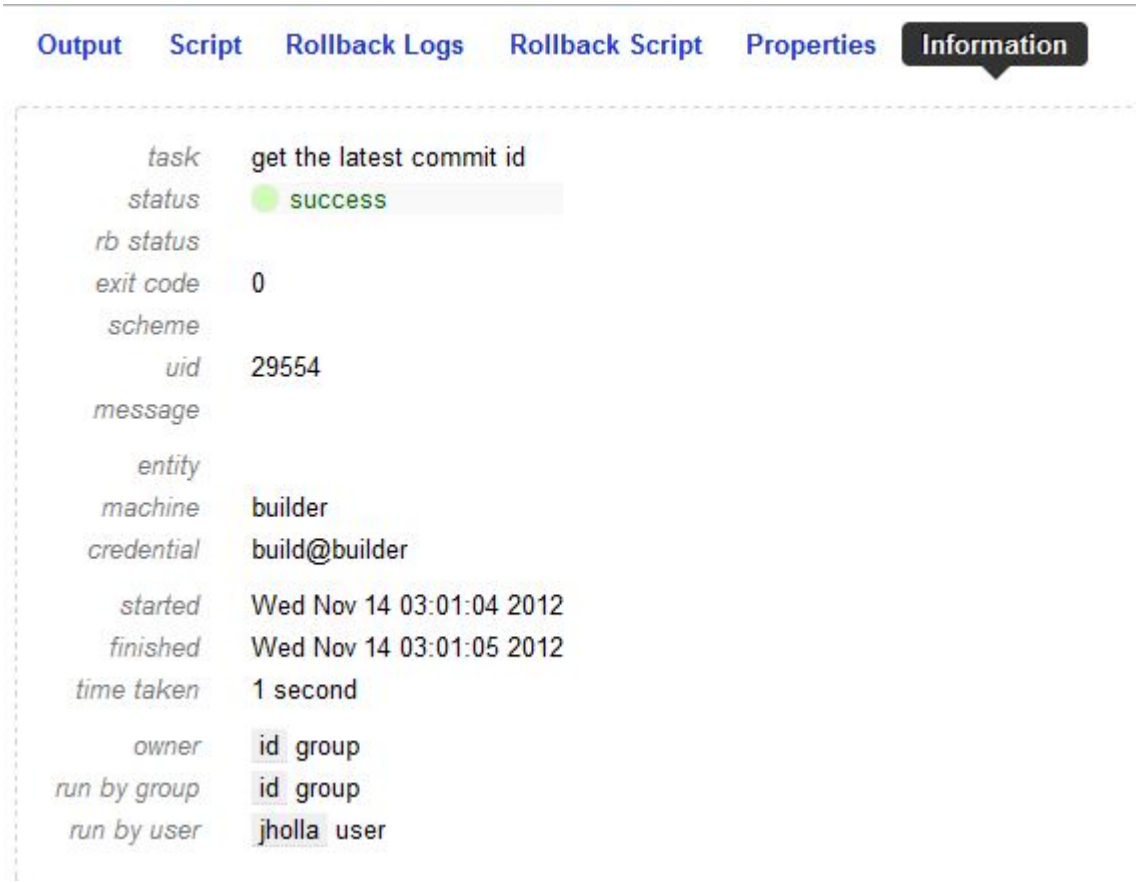

Figure 1.50: View task Information

All the child tasks and the scripts associated with the tasks are version controlled. The version information can be viewed in the version tab (figure 1.55). The older version can be activated by pressing the green arrow in the activate column.

1.14.3 Inactivating a Workflow

A workflow can be set to inactive in the edit workflow page. An inactive workflow can not be run or scheduled. An inactive workflow called from a "call workflow" task will error out.

1.14.4 Setting Pre conditions for a workflow run

In Epsilon it is possible to set restrictions on when can a workflow be run, whether to allow the rerun of a workflow, and if multiple instances of the workflow can be run. Using the tab "Pre-Conditions" in the workflow edit screen (figure 1.56) it is possible to allow Multiple runs of the workflow at once, and allow reruns of the workflow from the reports tab. It is also possible to specify the time, weekday, day of month and month between which the work is allowed or not allowed to run.

1.14.5 Generate a process document

Epsilon workflow also a live documentation of your business process. Unlike a offline word document this process is never dated. A printable copy of this process can be generated using

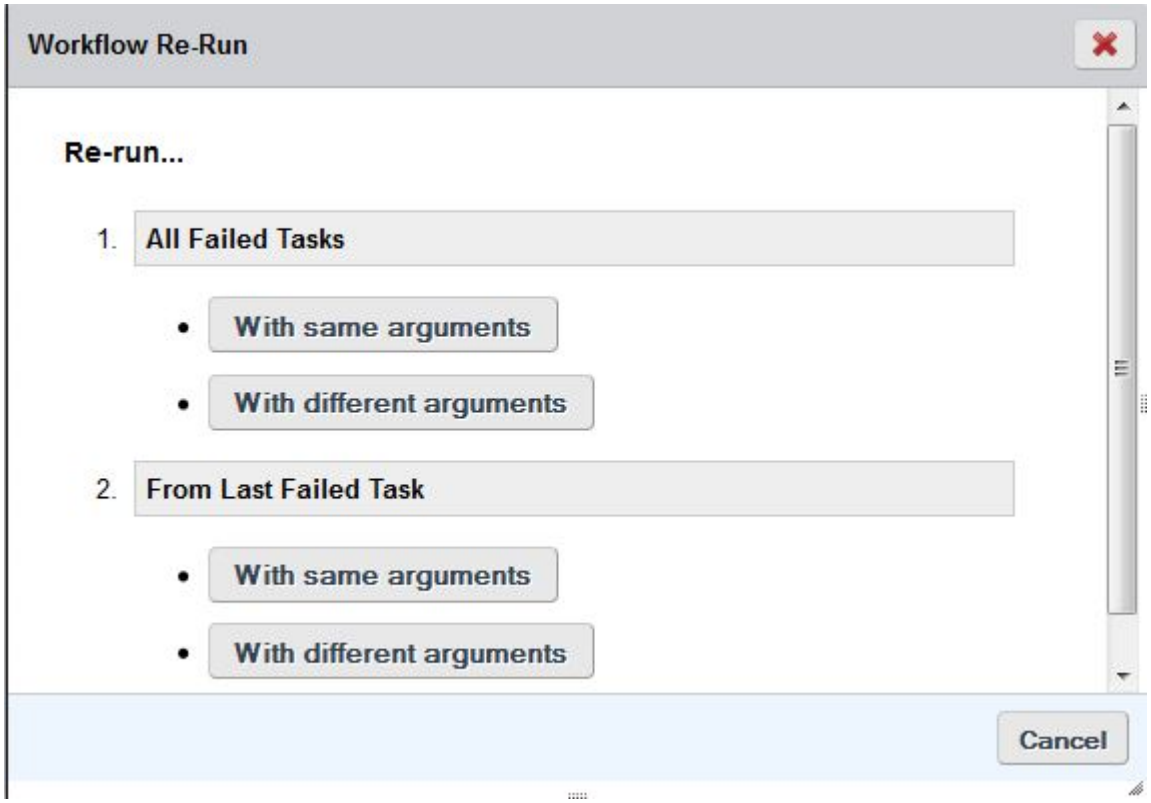

Figure 1.51: Re Run a Workflow

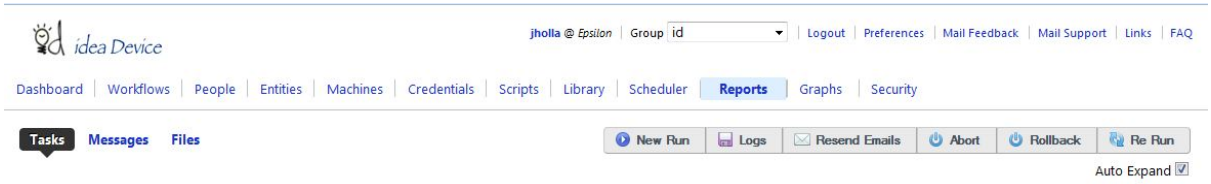
idea Device
jholla @ Epsilon | Group id | Logout | Preferences | Mail Feedback | Mail Support | Links | FAQ
Dashboard | Workflows | People | Entities | Machines | Credentials | Scripts | Library | Scheduler | Reports | Graphs | Security
Tasks | Messages | Files
New Run | Logs | Resend Emails | Abort | Rollback | Re Run
Auto Expand

Figure 1.52: Resend emails of approval and manual tasks

the "Document" button from the workflow edit page. This would generate a file in HTML format, detail of each and every task, which constitutes the current process of the customer.

1.14.6 Setting a workflow as Self Service Application

Epsilon has a unique feature by which a workflow owner can publish his workflow to another group. The recipient will only be able to run the workflow but will not be able to edit it. The recipient group can also use the workflow as part of their larger workflow. The workflow will run with privileges of the original group.

The SSA feature allows administrators to securely provide privileged access to other users. For eg. An Oracle Database Administrator can write a Oracle DB restart workflow, using well tested scripts in the correct order. The DBA can set the correct username, password and machine details in the workflow. This workflow can be published to a user group via the UI. The user group now gets the capability to restart a specific oracle db on demand without knowing the details of how to start the DB and without knowing the username and password for the database. The dba can edit this workflow and add an approval task as the first step. All future

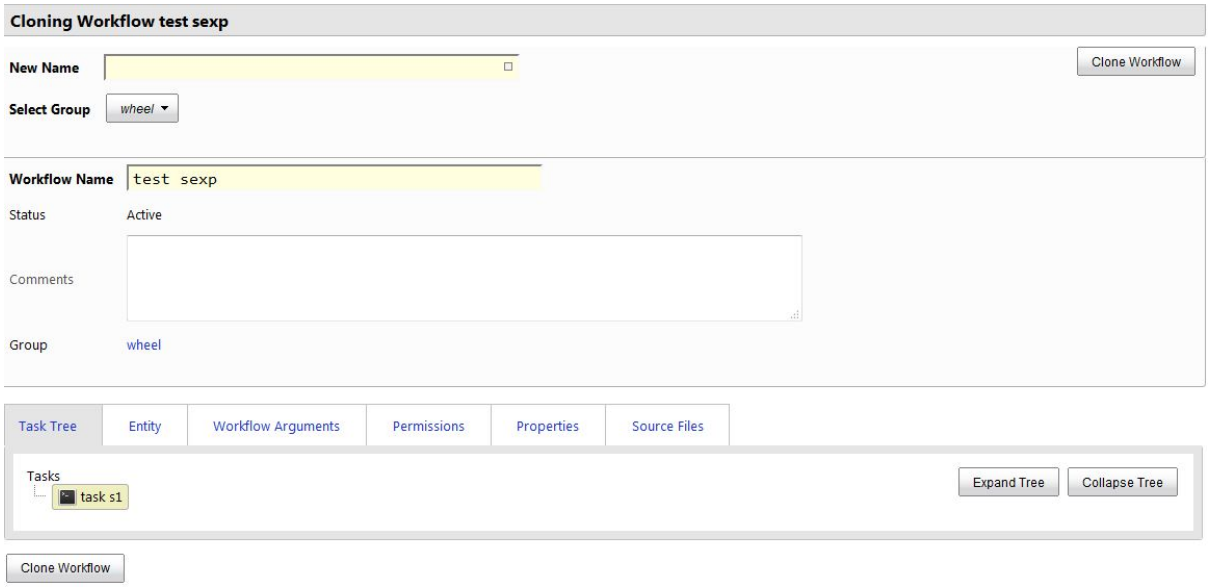

Figure 1.53: Clone a workflow

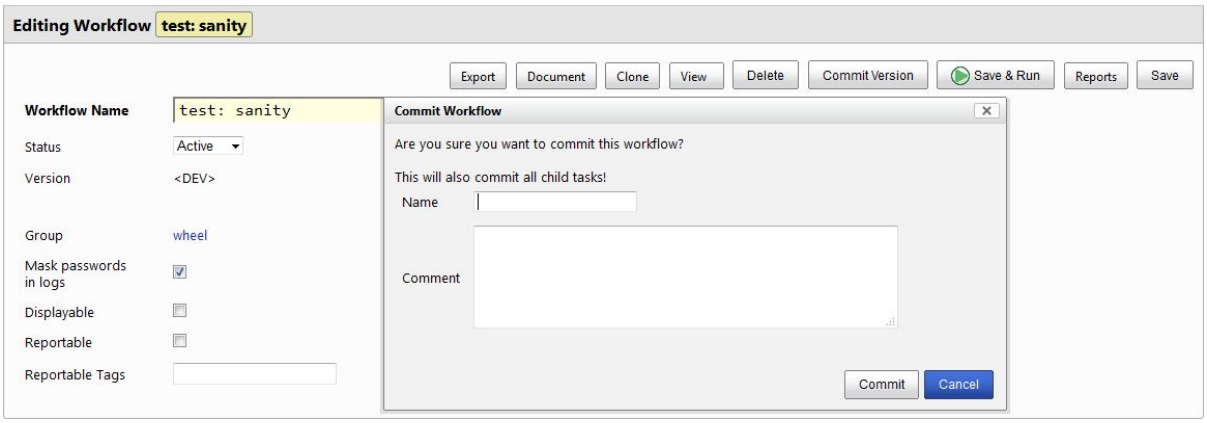

Figure 1.54: Commit a version

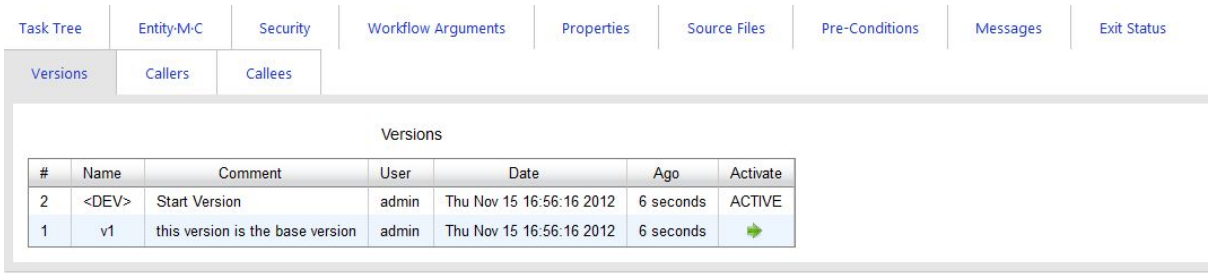

Figure 1.55: View and activate a version

executions of this workflow by the user group will require explicit approval by the DBA.

The security tab in the workflow edit screen has the relevant controls (figure 1.57)

You can control if the SSA user can see the individual logs of each task in the workflow, or if the user can only see the final output of the workflow or even if the user can see what tasks were run as part of the workflow. If you have a very sensitive workflow, you can just uncheck

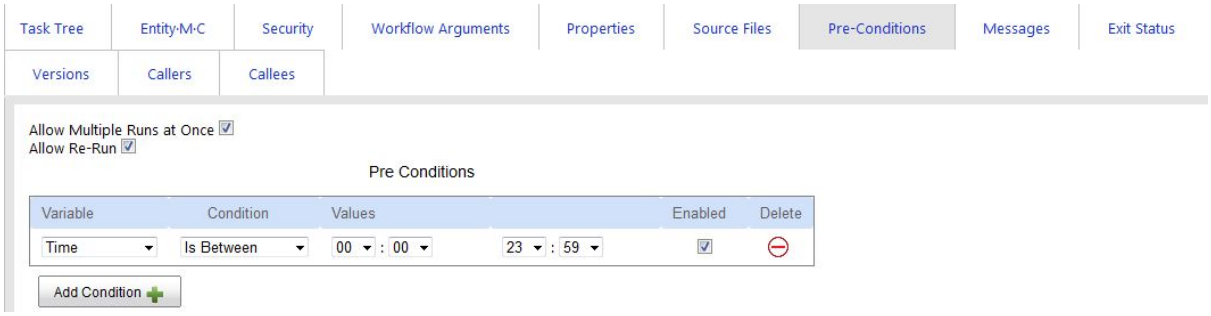

Figure 1.56: Setting conditions to a workflow run

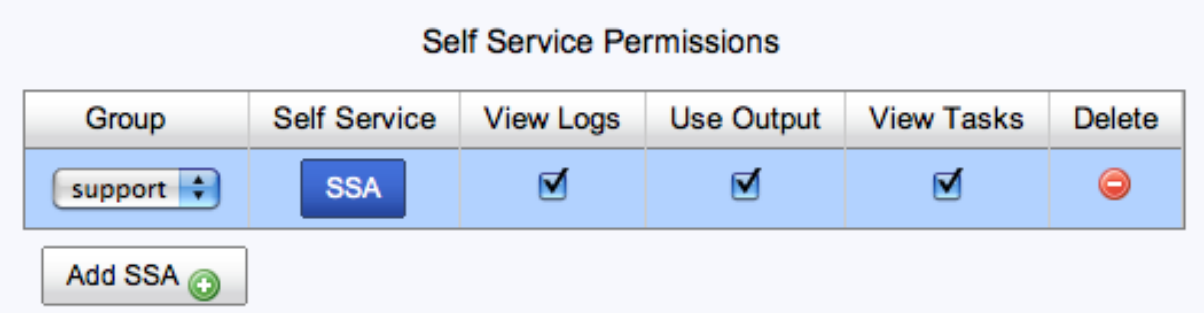

Figure 1.57: Enabling SSA

View Logs and View Tasks to prevent the user from seeing any tasks and their logs.

Chapter 2

Tasks

A task is one of the many steps that make up a process. For e.g., the process of bouncing a database will consist of 2 high level steps Stopping the database and Starting the database. Each of these steps maybe broken into smaller steps, each will be treated as a task.

In Epsilon, a series of related tasks make up a Workflow There is no requirement that each task must be a single functional step, at the user's discretion a single task can do many functional steps or may only do part of a single functional step.

Task Types For enhanced design experience, Epsilon offers a variety of task types.

2.1 Run Shell

A run-shell task can be used to run user scripts on a remote machine. The script can be in any programming language the remote machine supports.

Each run shell task has a script associated with it. One or more tasks may be using a script. If the script task is 'Epsilon Script', the run shell task will not connect to any machine and directly provide the script contents as the output.

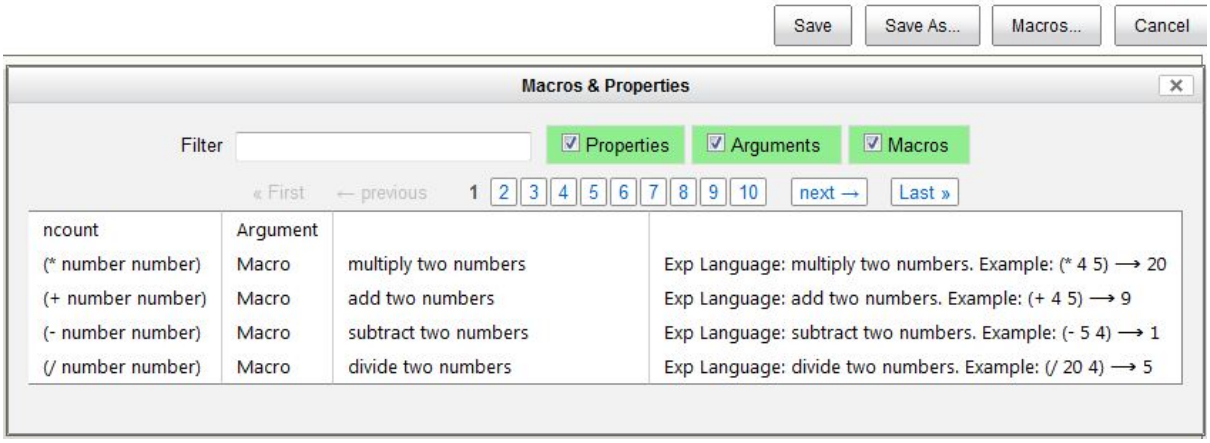

Figure 2.1: Macro Input

Scripts may be reused in multiple tasks. While updating a script used by multiple tasks, you may either decide to update the script or clone it. In case you update it, all tasks using

this script will now use the updated version. In case you decide to clone it, the task in question will use the cloned script whereas other tasks will keep using the original script.

While editing the script, you may have a look at the available macros by clicking on the "Macros" button (figure 2.1). This provides a list of Macros with dynamic search. You can select a Macro by clicking on it.

2.1.1 Passing arguments to a script

All values supplied in the Command Line Arguments textbox (in the Run Shell "Advanced" section) are supplied as command line arguments to the script in that task. T o supply multiple arguments separate each argument by spaces. To supply an argument with spaces in it, quote the argument like this: `\"argument with space\"`.

Example: arg1 arg2 `\"argument three\"` arg4 arg5 @@AZ_TASK_NAME@@ $HOME

In this example the script receives the following arguments in order: arg1, arg2, argument three, arg4, arg5, `some_task, /home/a_user`

2.2 Call workflow

You may call other workflows from your workflow. You can use Call Workflow task or Dynamic task for this.

You may chose the workflow to be called by clicking on the "Select Workflow" button.

Call workflow task will show the arguments for the selected workflow.If you do not provide the required arguments, Call workflow task will not be created.

2.3 Input

An Input Task allows you to ask the user running the workflow for input values or selections through the browser UI. Input tasks can render text boxes with defaults, checkboxes, file uploads, single selections and multiple selections. Input task stores the values into properties named by task output. Input task can run shell scripts or use Static text to feed custom UI into Input task output.

Input task syntax is:

```
propertyname1,type,label1="value0"
propertyname2,type,label2=["value1","value2","value3"]
```

propertyname1 is the name of the property for the value to be stored in. label is the display label to be shown to the user while inputting this value. label is optional, and defaults to the propertyname. type is a standard keyword, and can be one of **text, password, checkbox, select, selectmultiple, file, and message**. type is optional and defaults to text. The different values fr type are

- **text** Draw an input box with optional default value. The value of the property specified in first argument will be set to the text entered by the user. The double quotes " are necessary for text values.
- **password** Draw an input box with optional default value for password/secret input. The value of the property specified in first argument will be set to the text entered by the user. The double quotes " are necessary for text values.

- **checkbox** Draw a checkbox with the one value supplied as argument. The value of property with be value when checked, not set when (equivalent to empty) when unchecked. The double quotes " are necessary for text values. If the checkbox is selected the property contains the values assigned to it. For an unselected checkbox the value is empty.

- **select** Any multi valued property is exposed as a select. [and] are used to do grouping for multiple values. This shell script will generate this output. Please note that the strings need to be quoted, and the shell (and other languages) will need the " to be quoted by

- **selectmultiple** Multi select allows the user to select multiple values. The multiple values are stored as comma separated values in the property.

- **file**: draws a file upload form control. This allows the user running the workflow to upload a file to the workflow and have it available to all workflow tasks. The file is stored in the epsilon server and made available to via a special macro `@@AZ_PROPERTYFILE_propertyname@@`. In the following example, the file is made available to all scripts as `@@AZ_PROPERTYFILE_my_csv@@`. If this property is used in a task, it expands to the location of the file on the machine the task is running.

When the script type is epsilon script, the input task, coded as following, will yield a input dialogue box as shown in figure 2.2

```
 message,markdown="## Please Enter User Details"
 name,text,User Name="test123"
age,text,Age=""
new_password,password,Enter Password=""
new_password_repeat,password,Repeat the Password=""
reset_password,checkbox,Reset Password="yes"
my_users,select,Please select a user=["user1", "user2", "user3"]
my_values,selectmultiple,Select one or more values=["abc", "def", "ghi"]
my_csv,file,CSV File=""
```

In the first line, `message,markdown="## Please Enter User Details"`, the property name is message and the property value is displayed as header text in the input message box (figure 2.2). Setting type to markdown will process the property value through markdown processor.

Using HTML tags in the input task HTML tags can be used to format the dialogue box. When using the HTML tags, "Markdown Wiki Syntax" flag has to be checked.

```
<table>
<tr><td>
Please Enter User Details</td><td><input type=text value="123" name=age>
</td></tr>
<tr><td>
User Name</td><td><input name=name type=text value="test123">
</td></tr>
<tr><td>
Age</td><td><input name=age type=text value="">
</td></tr>
<tr><td>
Enter Password</td><td><input name=new_password type=password value="">
</td></tr>
<tr><td>
Repeat the Password</td><td><input name=new_password_repeat type=password value="">
</td></tr>
<tr><td>
Reset Password</td><td><input name=reset_password type=checkbox value="yes">
```

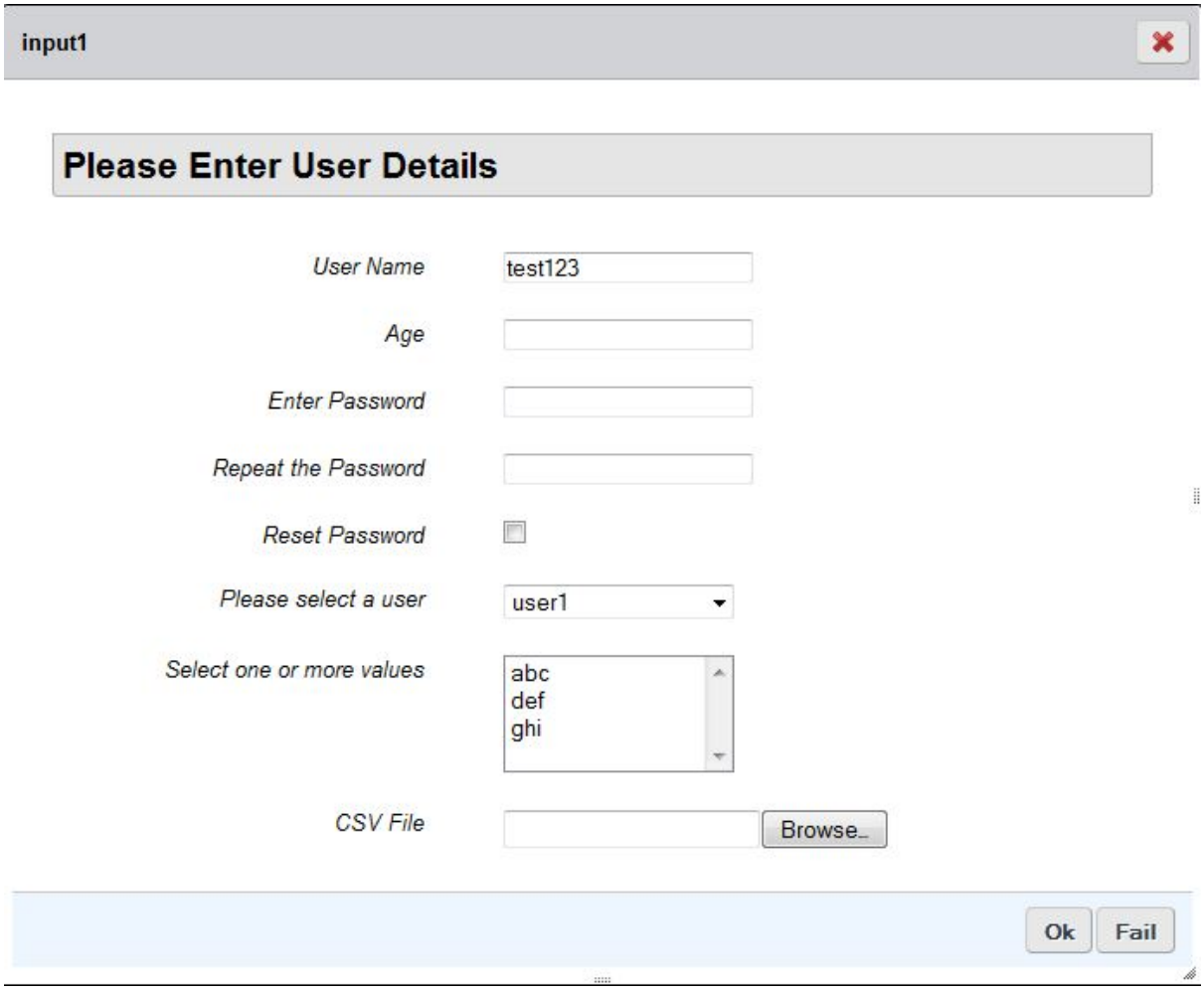

Figure 2.2: Input Task

```
</td></tr>
<tr><td>
Please select a user</td><td><select name="my_users">
                                        <option value="user1">user1</option>
                                        <option value="user2">user2</option>
</select>
</td></tr>
<tr><td>
Select one or more values</td><td><select multiple="multiple" name="my_values">
                    <option value="abc">abc</option>
                    <option value="def">def</option>
                    <option value="ghi">ghi</option>
                </select>
</td></tr>
<tr><td>
CSV File</td><td><input name=my_csv type=file value="" >
</td></tr>
</table>
```

Running other commands in the input task There may be a need to do other processing, for example run shell commands in the input task before prompting for the value. If the script type is "shell", the syntax for the input task is as given below. Remember that a "shell" type requires to be executed on a target machine, and hence would require a value for machine and credential.

```
echo name,text,User Name=\"test123\"
echo age,text,Age=\"\"
echo new_password,password,Password=\"test123\"
echo new_password_repeat,password,Repeat Password=\"\"
echo reset_password,checkbox,Reset Password=\"yes\"
echo my_users,select,Please select a user=[\"user1\", \"user2\", \"user3\"]
echo my_values,selectmultiple,Select one or more values=[\"abc\", \"def\", \"ghi\"]
echo my_csv,file,CSV File=\"\"
```

One other way to do preprocessing is to have the script type as "epsilon script". The syntax for input tasks, while using "epsilon script" is

```
#script

users=["user1", "user2", "user3"]
print "message,markdown=\"## Please Enter User Details\""
print "name,text,User Name=\"test123\""
print "age,text,Age=\"\""
print "new_password,password,Enter Password=\"\""
print "new_password_repeat,password,Repeat the Password=\"\""
print "reset_password,checkbox,Reset Password=\"yes\""
print "my_users,select,Please select a user=%s" % (users)
print "my_values,selectmultiple,Select one or more values=[\"abc\", \"def\", \"ghi\"]"
print "my_csv,file,CSV File=\"\""
```

The syntax of "Epsilon Script" is derived of Python language.

2.4 Interactive

The interactive task itself is scripted as an "Epsilon Script"

Sample interactive task:

```
interactive
p = _connect(cmd='rm -i /tmp/foo', shell='bash')
i = p.expect(['remove.*\?', '.*password.*', TIMEOUT, EOF])
if (i 0) {
p.sendline('y')
}
elif (i 1) {
print "bad password"
}
else {
print "command failed %s" % str(p.match)
}
p.close()
exit(p.exitstatus)
```

_connect() opens a ssh connection to the current EMC and optionally executes a program which can then be interacted with using standard pexpect apis. The pexpect module itself is available as the symbol '_P'.

Epsilon script itself is whitespace insensitive - using braces for indentation, supports string interpolation and provides a restricted set of python builtins (no imports allowed). The 're' module and the 'Exception' class symbols are made available. Limited command substitution (`-backtick operator) support is available. Epsilon script parser also understands the current line based static setup and is backward compatible with existing deployments.

2.5 Approval

In case you need approval from one or more persons before a task or set of tasks can be performed, you may add an Approval task. You can attach a message along with the approval request.

You can also assign an optional time-out. The Approval task will return with failure if approver does not respond within the time out period.

The Approval task will send out Approval Request to these people and meanwhile pause the Work- flow. The workflow will resume when:

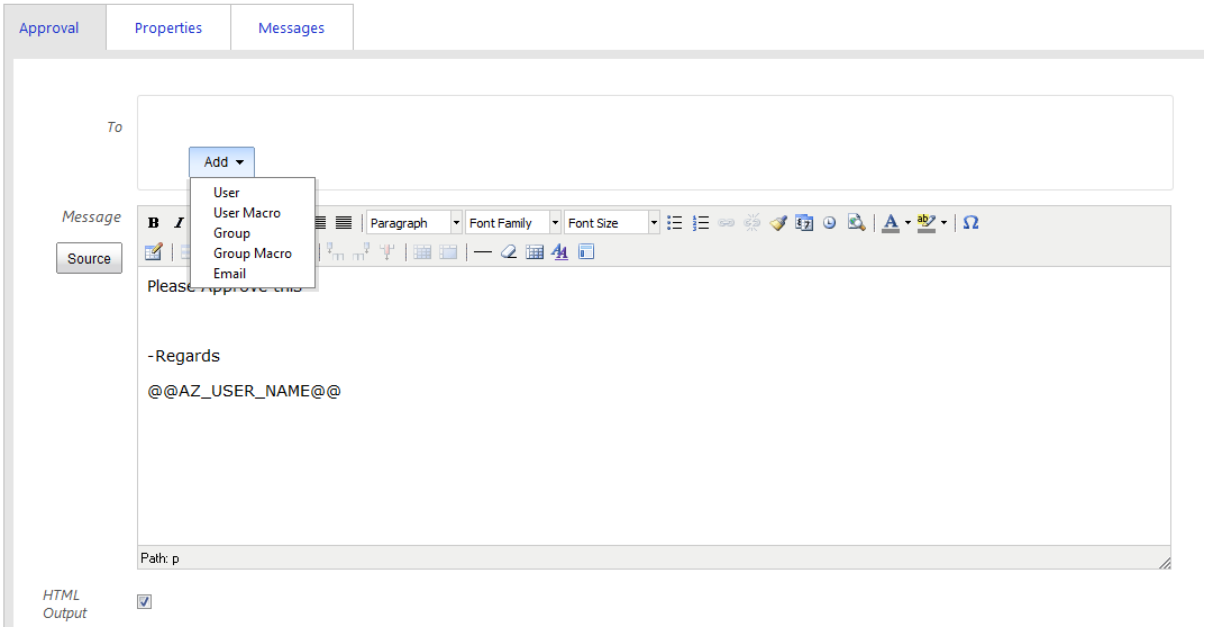

Figure 2.3: Basic Approval task

- Approver approves or rejects the request.
- Time-out happens. This will cause Workflow to Fail.

Figure 2.3 shows that the approval target could be, group, Epsilon users or email id of the users.

The approval body may have some inbuilt macros such as username who ran this workflow (@@AZ_USER_NAME@@). You can also place html tags in the body, which would be rendered properly when viewed in a html supporting email client. Be sure to check the HTML Output check box, if you have used html objects such as tables or lists in your email body.

You may add one or more Approvers. When there are more than one approvers, a polling can be set where you can specify the minimum number of approvals or rejects required for succeeding the task.

Figure 2.4 shows how to input Approval Count and Reject Count in an approval task by clicking on "Show Advanced" button.

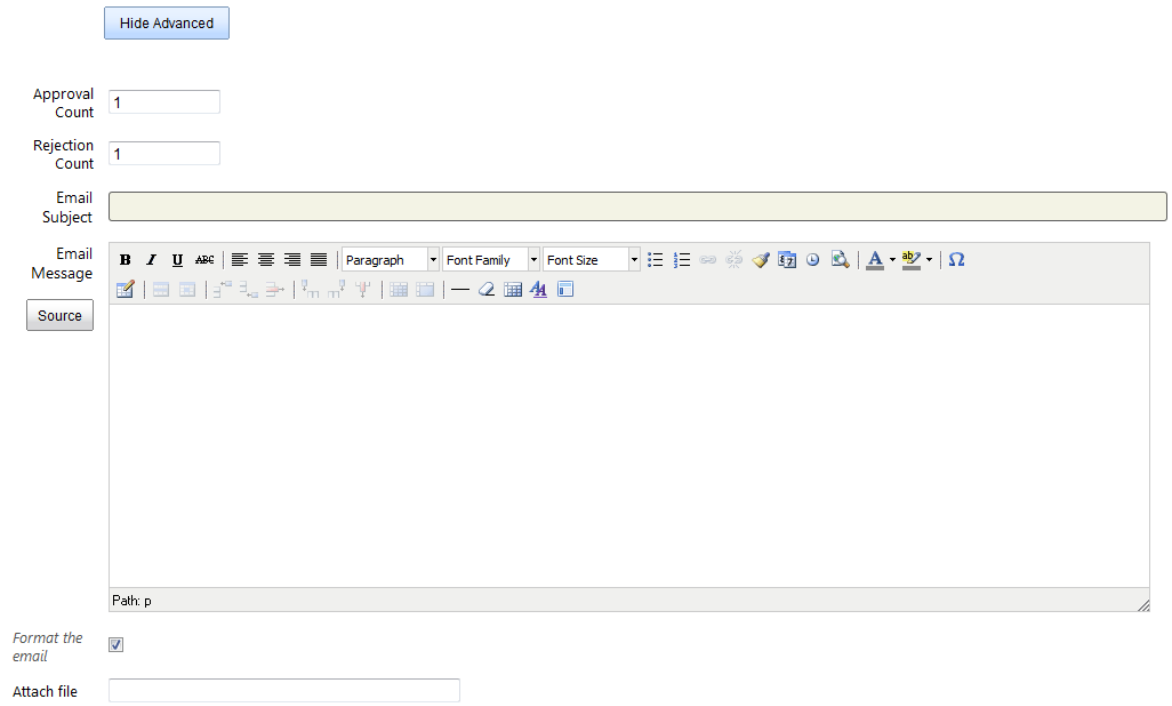

Figure 2.4: Advanced tab of Approval task

Customizing the approval message : The email message sent as a part of approval task can be formatted by entering the e-mail subject and the message body present in the "Advanced" tab. The text entered in "message" field shown in figure 2.3 will not be part of the email message if email body in the advanced tab is filled.

Advanced Customization of approval mail The approval mail body can be further customized,exactly to your need by uncheking the checkbox "Format the Email" shown in figure 2.4. this check box is available in the advanced section of the task. To approve a task it is important to send a url which facilitates the approver to approve or reject. This url is available as @@AZ_APPROVAL_URL@@. This token should be part of the message body when you generate your custom email body by unchecking the checkbox "Format the email". The other tags which may be of interest are

@@AZ_APPROVAL_STATUS_ID@@ The status id for current approval task.

@@AZ_APPROVAL_URL_TOKEN@@ URL token for current approval task.

@@AZ_APPROVAL_AUTH_TOKEN@@ Auth token for current approval task.

@@AZ_APPROVAL_TARGET@@ Target at which the current approval is targeted.

Synchronous approvals An approver may approve by clicking on the url embedded in the email. A sample message, with url is shown below

```
Please approve or deny this request at
http://rel/approval/decision/3/2?auth=TIkTw9dVvdWetRiJFiMSoS8OmJQ%3D
```

Approver, optionally may be required to log in before approving. It is possible to accept certain inputs from the approver or allow approver to overwrite certain inputs.

Asynchronous approvals Asynchronous approvals can be triggered by replying to the approval mail or clicking on a link embedded in the email, which opens the email client to send a mail. To achieve this you have to configure fetchmail, set up a mail box to receive the approval mail and configure the workflow "Dispatch Reply Mail for SD" in the group "Service Desk".

Resending the Approval Mails : The resend emails button in the reports tab can resend the emails to the recipients of an approval task(figure 1.52).

2.6 Decision

Decision task provides the flexibility of running one of the children tasks based on the output of the parent task.

Decision task performs two actions:

- Execute a script
- Run a child based on the script exit code. If the script exits with exit code 0, the 0th child is run and so on.

For example if the business logic requires that you want to execute a task called "child 1" if a variable called count is set to 1, execute task "child 2" if the variable count is set to 2, else take no action and just continue the execution of the rest of the workflow. Using a shell script, such a flow can be define as shown in figure 2.5. The exit code returned by @@AZ_NOOP@@ tells the decision task to exit without taking any action.

The same action can be coded using "epsilon script" as

```
#script
if "@@count@@" == "1":
 exit(0)
elif  "@@count@@" == 2 :
 exit(1)
else:
 exit(@@AZ_NOOP@@)
```

Note: To execute multiple tasks under a given decision node, create a meta task and create the sub tasks under this meta task. For example in the above example if for exit(0) you want to execute 2 tasks t1 and t2, then create a meta task as the first node and create t1 and t2 under this meta.

2.7 Parallel

All children of parallel task will run in-parallel. This is highly beneficial in scenarios where in same action has to be performed multiple times with no inter-dependencies.

The screen shot in figure 2.6 shows how to run 2 tasks w1.t1.1 and w1.t1.2 in parallel. Create a task w1.t1 of type parallel and create 2 sub tasks which have to run in parallel.

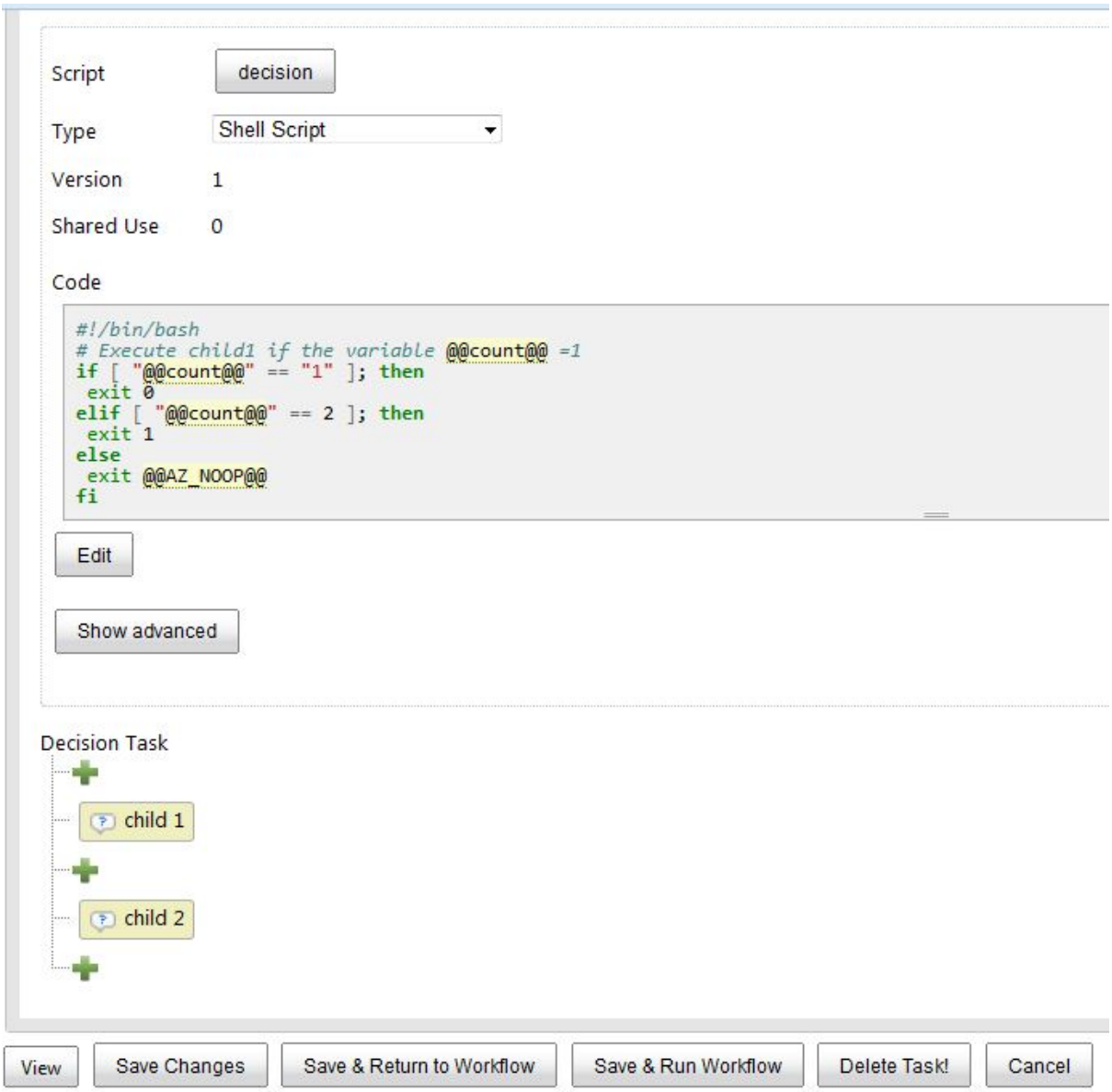

Figure 2.5: Defining a decision task

2.8 Meta

You may club one or more tasks together under a meta task. You may add as many children tasks as required. Meta tasks are useful when there is a need to execute multiple tasks under a loop task, parallel tasks, and a decision node.

For example, if you want to run 2 tasks serially, but in parallel with a 3rd task, group the 2 tasks under a meta task and keep the meta and the 3rd task under a parallel task. This setup is shown in figure 2.7.

2.9 Notification

Send out notification to Epsilon user or to an email address.

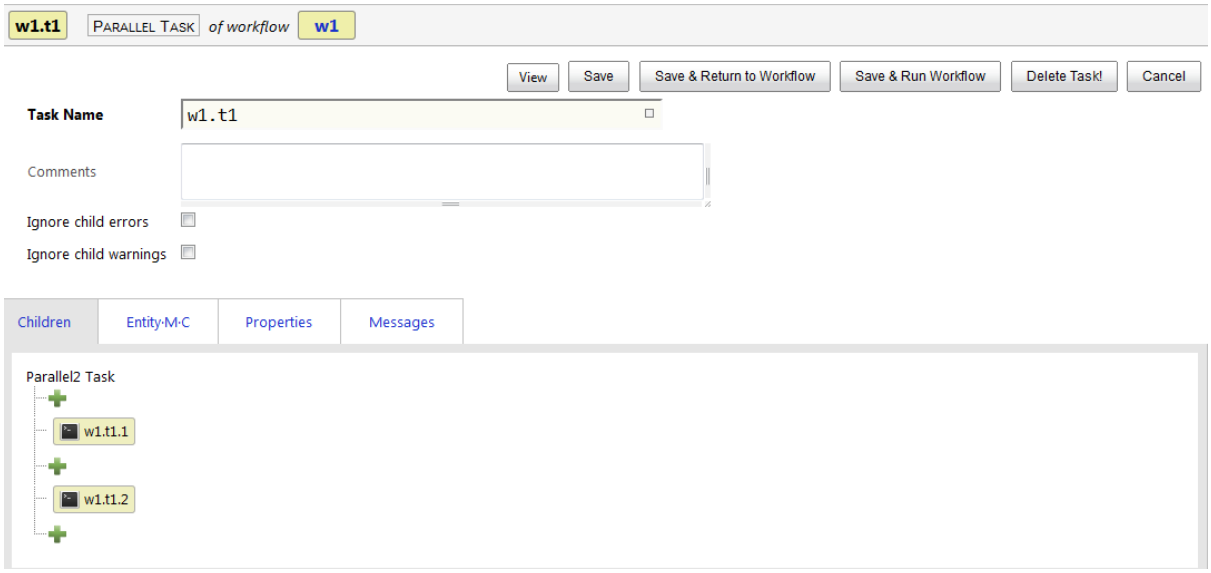

Figure 2.6: An Example of Parallel task

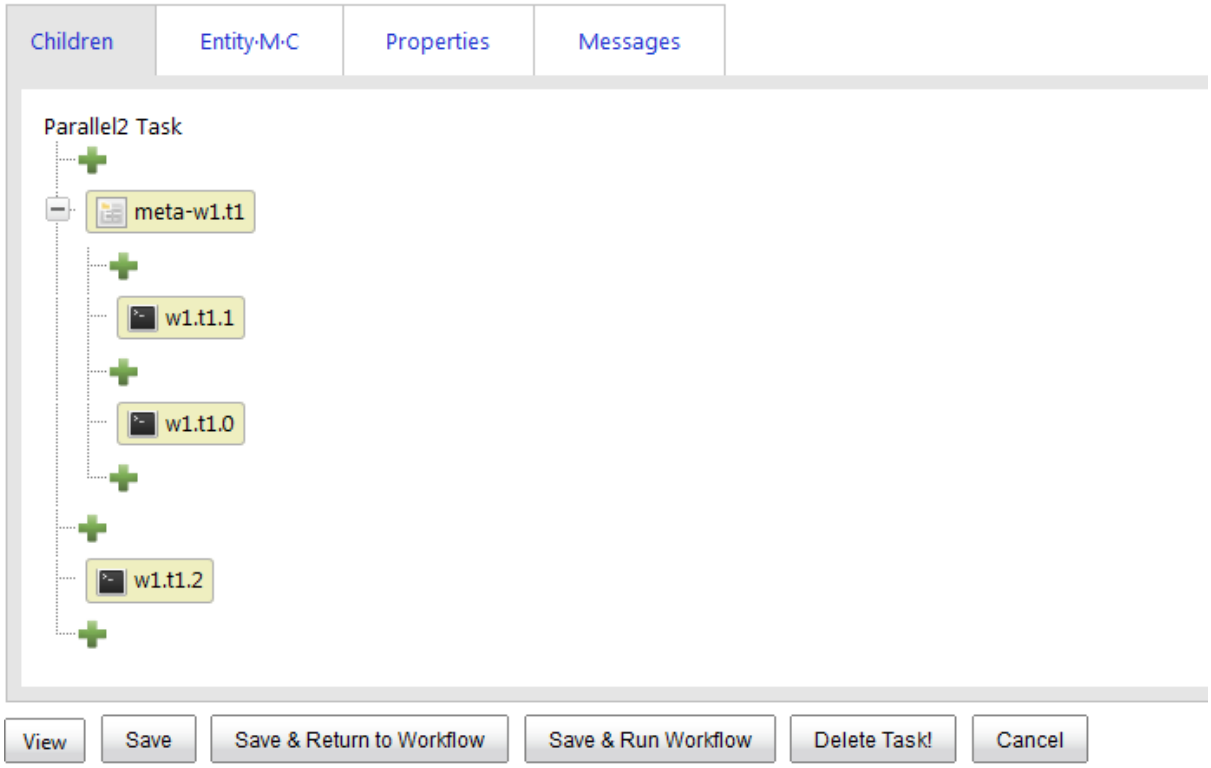

Figure 2.7: An Example of Meta task in Parallel

2.10 Email

Send out an email to the address provided. Email task supports sending attachments as well. Further an html message can be sent by checking the flag "Markdown Wiki Syntax"

2.11 Manual

In case you want a task to be done manually by a particular person, you may send out a Manual request to that person using Manual task.

You may either specify an epsilon user or an email address to which the request is to be sent.Please not that you can specify only one user or email address per Manual task.

Manual task also lets you specify Instructions for the User.

You may also associate a time-out period with the task. If there is no response within that period, the Manual task will Fail.

2.12 Delay

A delay task introduces a certain delay before the workflow proceeds.

2.13 Eval

An Eval task lets you set new properties on run-time. You may use Eval task to modify already existing properties as well.

Please note that the scope of Properties set or modified by Eval task is limited to that very run of the Workflow.

The scope of property set/edited by an Eval task is limited to only it's current branch. In case you want to set/reset property for the entire Workflow, you may turn on the global flag. To turn on/'off the global flag, click on "Advanced" and check the Global Flag.

The syntax for eval task is alpha=beta sets a property alpha with value beta. Suppose gamma=beta, alpha=@@gamma@@ sets a property alpha with value as expanded value of property gamma.

@@theta@@=omega will set a property with name as expanded value of "theta" as value omega.

`@@lambda@@=@@epsilon@@` will set a property with name as expanded value of "lambda" and value as expanded value of "epsilon"

The syntax, while using script type "shell script" is (requires connection to a machine)

```
echo alpha=@@gamma@@
```

The syntax, while using "epsilon script" (does not require connection to a machine) is

```
#script
print "alpha=%s" %( @@gamma@@ )
```

2.14 Dynamic

Dynamic task is used to call a workflow at run-time.

The syntax for dynamic task is:

`WORKFLOW_NAME,WANT_SUCCESS,PARAM1=VAL1,PARAM2=VAL2,PARAM3=VAL,....`

where

`WORKFLOW_NAME` is the name of the workflow to be executed

`WANT_SUCCESS` means the workflow should succeed, any other string there means we do not care about the status.

PARAM1=VAL1 placeholders for workflow arguments if any

You can have multiple such lines and the dynamic task will execute each of them. You can generate the lines dynamically running a script.

Please note that if all required arguments are not supplied, Dynamic task will Fail. To call the same workflow more than once, you may use Loop task as well.

2.15 Update

You may wish to update a Credential or Property at run-time. Update task allows you to do so.

The syntax for update task is

```
PROPERTY^^<workflow_name>^^WORKFLOW^^secret_<yes/no>^^<property_name>^^<new prop value> PROPERTY^^<workflow_name>^^T
ASK^^<task_name>^^secret_<yes/no>^^<property_name>^^ <new prop value>
CREDENTIAL_UID^^<credential_name>^^<new_value> CREDENTIAL_P ASSWORD^^<credential_name>^^<new_value>
MACHINE^^<machine_name>^^PROPERTY^^secret_<yes/no>^^<property_name>^^<new prop value>
ENTITY^^<entity_name>^^PROPERTY^^secret_<yes/no>^^<property_name>^^<new prop value>
```

2.15.1 Rollback for Update task

Update task maintains the state information internally. In case of rollback, the previous values are restored. Users need not supply any rollback script for the same.

2.16 Confirm

In case you want a confirmation before performing a certain task or set of tasks, you may use Confirm task. There are three possible buttons shown in the confirm dialogue. The labels of the 3 buttons can be changed. The first button indicates a success, and the third button indicates a failure. The second button indicates that the workflow needs to stop.

2.17 Report

A report task can be used to generate wiki-formatted output.

2.18 Loop

To run a task in loop, you may use LOOP task. This provides for and while loop functionality through the UI. To execute multiple tasks in a loop, group them under a meta task.

2.18.1 While loop

In figure 2.8, the loop counter value is stored in the property name given in 'Loop Count Variable'. "Repeat Count" tells the number of times the loop needs to execute. The loop can exit either when the repeat count is reached, or an exit condition is achieved. The exit conditions can be a child of the loop task going into success, failure or warning status. It is also possible to ignore the child exit condition, then the loop would exit when repeat count is reached. It is possible to introduce a delay between each iteration using 'Loop Delay". It is also possible to introduce a time out to exit the loop if the time taken to execute the task exceeds a time limit.

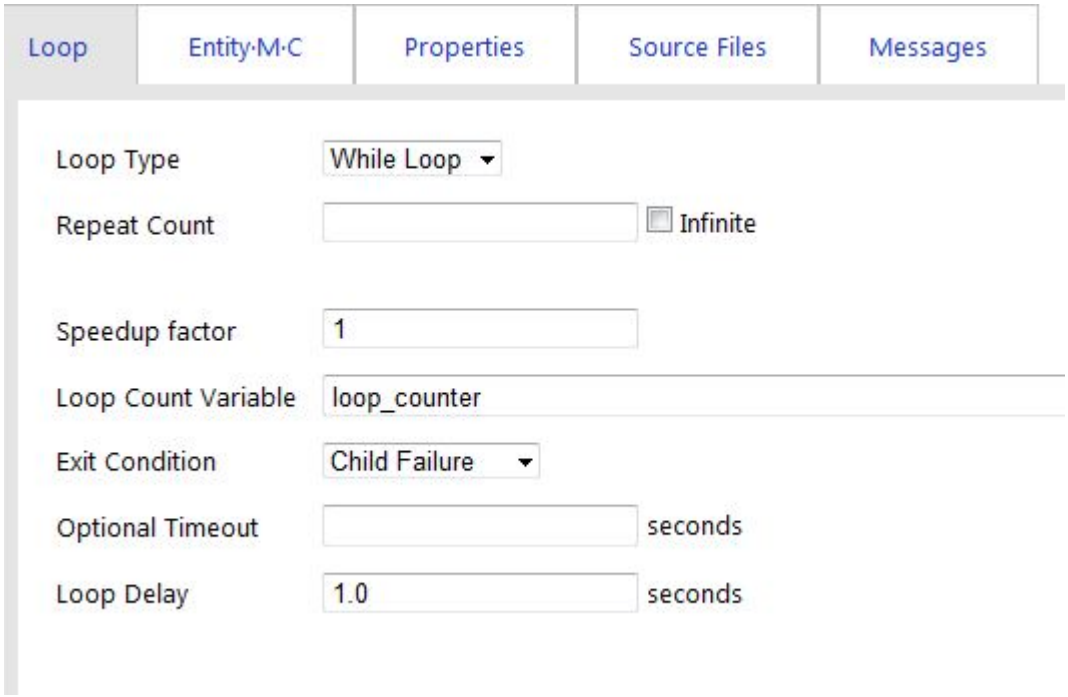

Figure 2.8: Defining a while loop

2.18.2 For Loop

For loop will loop over a list. This list is specified in 'Loop Over List'. The list is parsed based on the 'Loop List Delimiter'. The list variable would be available under property name specified in 'Loop Variable'. The loop counter is would be available under property name specified in 'Loop Counter Variable'.

For loop also supports the execution of multiple iterations in parallel for the values given in the 'Loop Over List'. This can be set by changing the value of 'Speedup Factor'. For example, by setting the 'Speedup Factor' to 5, five copies of the child tasks will be executed simultaneously for 5 different values from the 'Loop Over List', thus reducing the overall execution time of the workflow.

2.18.3 Loop Over Multiple Machines

. It is a common requirement to execute a workflow over a set of machines. This is easy to achieve in Epsilon. First create an entity and attach the required machine to this entity. Create a task of type loop. Select For loop as the loop type. For the loop over list we want to fetch all the machines associated to one entity, which is given at the runtime. This can be done using an inbuilt macro called `AZ_LIST_Entity`. We use THIS to indicate the current entity which has been set by the user. Hence
`@@AZ_LIST_Entity(name="THIS").get(Machine(name))@@` gives the list of machines delimited by ,. Loop variable `mch_name` holds one value of the machine for each iteration of the for loop. Speedup factor 1 indicates that each iteration is to be executed sequentially. A value greater than 1 would execute the iterations parallel. For example if Speedup factor=2 , two machines at a time will be simultaneously picked up for db startup (Figure 2.9)

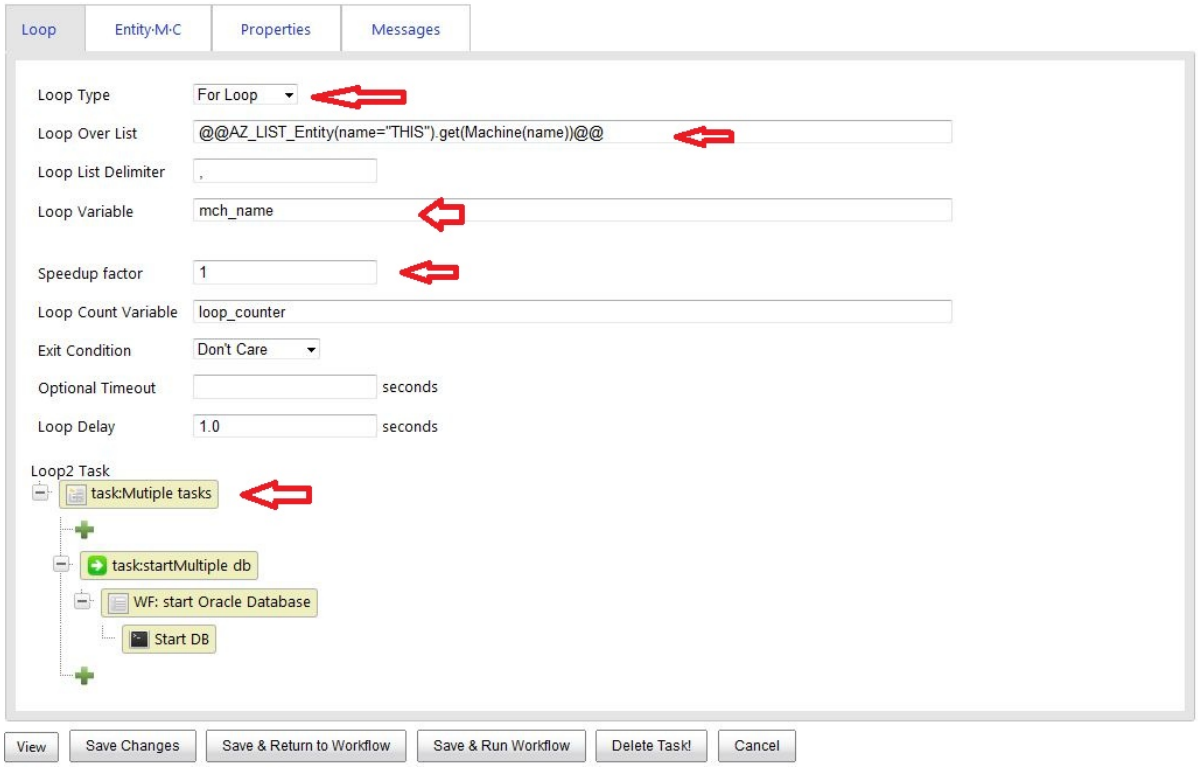

Figure 2.9: Loop Task

2.19 Method to Succeed or Fail a task

By default, a workflow stops whenever a task fails. The workflow continues running as long as the tasks are in SUCCESS or WARNING state. A parent task succeeds only if no child task fails. A parent task can be made to succeed by manipulating the exit codes.

There are 2 ways to SUCCEED or FAIL an Run Shell task.

- The first method uses the exit code of the script executed by the Run Shell task. In the exec task edit screen you can specify which exit codes mean the task has succeeded and which means the task has failed. The special exit code '*' means 'all other exit codes'. An example is shown in figure 2.10. In this example if the script exits with code '0', the task

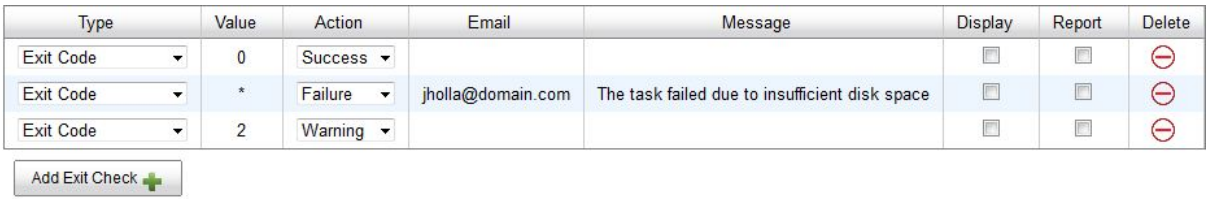

Figure 2.10: Simple Exit Codes

will be considered a SUCCESS, all other exit codes means the task has FAILED. This particular task will also email 'jholla@domain.com' on all non '0' tasks. You can setup as many exit codes as you want and each exit code can be configured to send email to any email id. It is also possible to set a success or failure message.

- The second method uses pattern matching in the output of the script. You can specify

any number of patterns and their associated actions, that will be matched against the output generated by the script. Important Note: Epsilon will always look at Exit Codes first to determine SUCCESS/FAILURE. Only after a task has been determined to have a successful exit code will Epsilon look for pattern matches - i.e - For pattern matching to happen, a task must have a SUCCESS exit code. An example is shown in figure 2.11 In

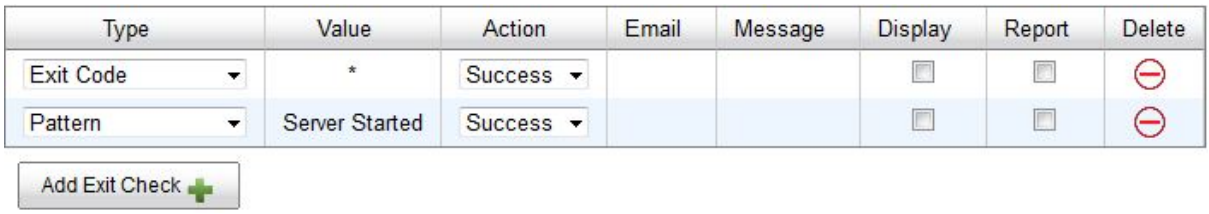

Figure 2.11: Simple Pattern Matching

this example, any exit code will always result in a SUCCESS. Epsilon will then search for the pattern 'Server started' within the output of the script. If this pattern is found, the task is marked as SUCCESS. If this pattern is not found, the task is marked as FAILURE. The pattern to match can also be a "Perl style" regular expression. Epsilon supports full Perl style and "grep -E" style extended regular expressions (figure 2.12). The following

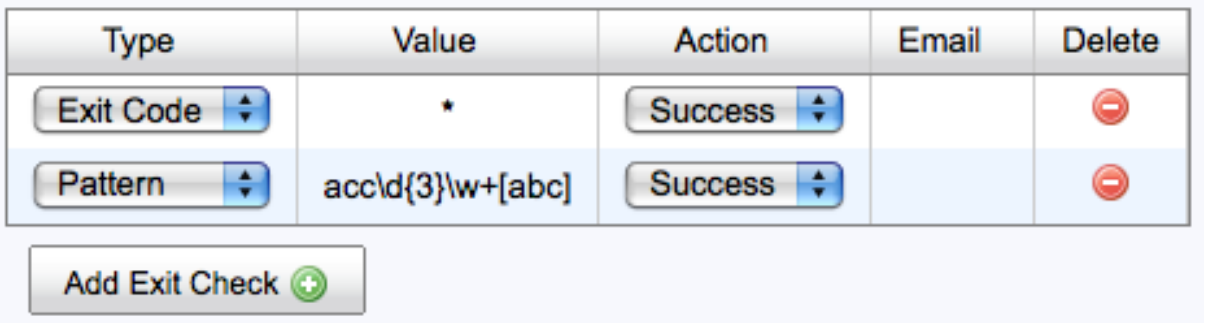

Figure 2.12: Pattern Matching with regular expression

rules apply when using pattern matching:

- Epsilon will always look at exit codes before looking for pattern matches
- Epsilon will always search for FAILURE patterns before searching for SUCCESS patterns
- Multiple patterns can be specified. Patterns can be simple words or perl style regular expressions
- If SUCCESS patterns are specified, all SUCCESS patterns must be matched for the task to SUCCEED
- If multiple FAILURE patterns are specified, any single match is enough for the task status to be set to FAILED
- If only FAILURE patterns are specified, a non-match with all of them is considered a SUCCESS

- There may be a need to stop the execution of a task immediately after finding a pattern in the log output. This can be achieved by setting Type as "Immediate Pattern" (figure 2.13). The rest of the rules of immediate pattern match is the same as Pattern match.

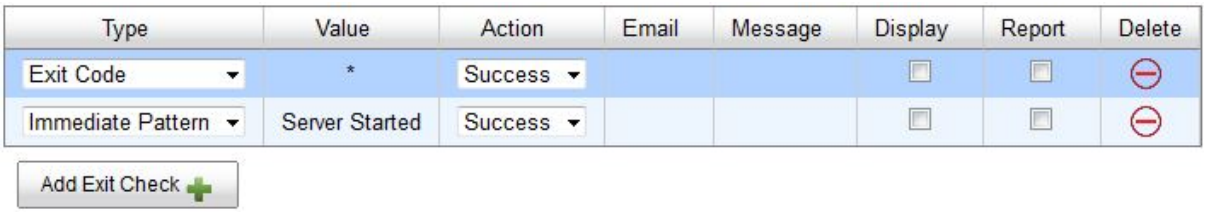

Figure 2.13: Immediate Pattern Matching

The success, error and warning messages can be further refined in the messages tab in the task edit page as shown in figure 2.14. The start message is displayed at the beginning of the task

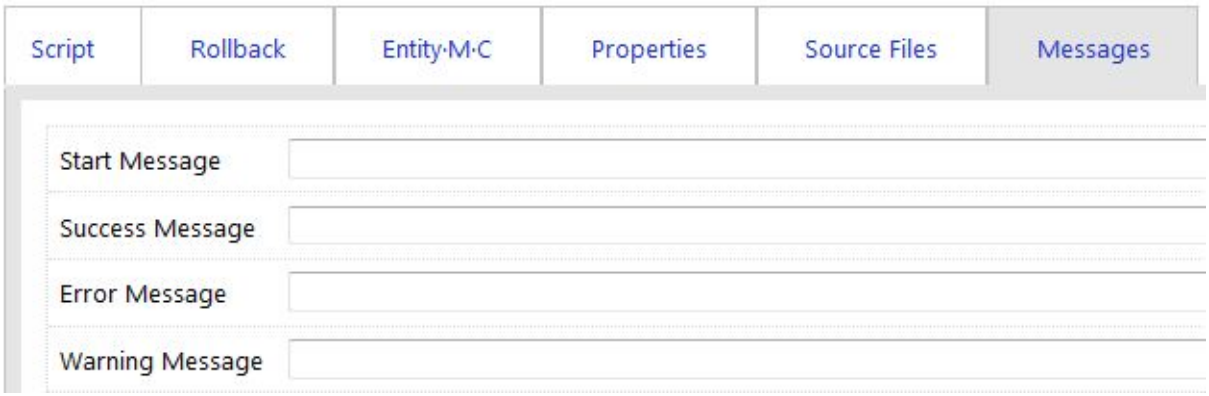

Figure 2.14: Immediate Pattern Matching

execution and the error, success, warning messages are displayed based on the final task status.

Chapter 3

Properties

A property is a key value pair created by a user or a group. This key value pair is used for macro expansion purpose. Examples of properties are: ORACLE_HOME=/u01/oracle, CONFIG_DIR=/etc/httpd/conf, TMP=/tmp, alpha=beta, a=1234

When Epsilon encounters a macro it will check the available properties of for name that matches the macro and if found, it will replace that macro with the value of the property. For eg the macro @@CONFIG_DIR@@ will be replaced by /etc/httpd/conf in the previous example.

Properties are owned by users or groups, a property therefore is specific to that user/group. A group cannot access the properties of another group except in special circumstances. A group can create as many properties as needed, different groups can create similarily named properties but they can only access their instance of the property. Properties are always associated with a task, workflow, or entity. In other words, properties are simple key value pairs set by a user/group on a task, workflow or entity. When a property is set on an entity, all workflows belonging to that group using that entity will have access to that property. Similarily all tasks beloning to a workflow will have access to all the properties set on that workflow by that group.

3.1 Property Inheritance

Property inheritance is the method by which tasks can access properties that are not directly set on them. Property inheritance is automatic within Epsilon, the user does not have to do anything special for inheritance to happen - but it is worthwhile to have an understanding of property inheritance.

There are 3 types of property inheritance in Epsilon.

- Entity based inheritance: This is a straight forward inheritance. If a group sets a property on an entity then all tasks and workflows run by that group on this entity will be able to access the property. This is how workflows & tasks can "inherit" properties set on an entity
- Workflow based inheritance: If a group sets a property on a workflow, then all tasks that are part of the workflow will automatically inherit that property. This is true for all child tasks. All children inherit the properties of their parent tasks.
- Group hierarchy base inheritance: This is a more contrived form of property inheritance. Groups are hierarchical, i.e groups can have a parent and many children. Epsilon will automatically provide all child groups with read-only access to the properties set by the parent group. Since group hierarchy is meant to model real world teams, group based

inheritance allows a senior group to perform one time setup of correct properties for workflows and entities which can be reused by all the junior groups.

There are 2 ways in which an inherited property can be overriden in Epsilon.

- Unsetting the property: The Epsilon UI allows you to select and unset an inherited property on a per task basis. Unsetting a property does 2 things (i) the property is removed from the set of available properties for that task & (ii) the property is not available to any children of that task By default, tasks will show the inherited properties. To unset any property, just delete the inherited property at task. You will now see the property value as '@AZ@_VALUE_UNSET'. Do not feel scared, this just means that the property will expand to '' in that particular task. To go back to using the workflow level property, you can delete the property with '@AZ@_VALUE_UNSET' value at the task level.

- Overriding the property: The value of an inherited property can be updated at the task level. The Epsilon UI can be used to update the value of an inherited property. This new value will be available to the task and it's children.

Chapter 4

Macros

A Macro is a special token that is expanded at run time by Epsilon. Macros can occur in scripts, templates, command line arguments, in email addresses, subject and body and usually any place that text can be entered. An example of a macro usage is :

```
cp daily.log /var/@@BACKUP@@/
```

@@BACKUP@@ is a macro. All macros are of the format `@@<name>@@`. When Epsilon encounters a macro, it will replace the macro with the appropriate expansion. Macros are used to make scripts generic and are the basis of creating reusable workflows.

Epsilon expands macros by looking at 4 sources.

- Properties: A property is a key value string that is set by a user. When a user executes a workflow containing macros, the user's properties are examined for a property with the same name as the macro, the macro is then replaced with the value of that property. For eg, if the user has a property BACKUP=bkdir, then the macro @@BACKUP@@ is replaced with the string bkdir, and the script

```
cp daily.log /var/@@BACKUP@@/
```

  becomes

```
cp daily.log /var/bkdir/
```

  Since properties can be set per user or per group, the scripts are expanded appropriately for each group/user. This allows generic scripts to be used by multiple users/groups.

- Workflow Arguments: Epsilon workflows can define runtime arguments. The value for these arguments are supplied by the user on the workflow run screen. For eg, a workflow may define BACKUP as a runtime argument. When a user starts this workflow, Epsilon will query the user for a value for BACKUP. Any script within this workflow containing the @@BACKUP@@ macro will have that macro expanded to the value supplied by the user at runtime.

- Builtin Macros: Epsilon supports a large number of builtin macros. The user does not have to set properties or workflow arguments for the macros to be expanded. Refer the faq for the list of Built-in Macros

- EVAL Tasks: This is an advanced usage scenario. The output from eval tasks is used to dynamically update the values of properties for a workflow. Because macros are expanded by looking at properties, EVAL tasks can be used to compute values for macros at runtime.

4.1 Macros to access the output of previous tasks

Epsilon allows a task to access the output of multiple previous tasks even if they were executed on different machines. There are 6 ways to access the output of another task in the current task.

Using the macro `@@AZ_TASKOUTPUT_PREVIOUS_<N>_NOSTRIP@@`. This macro expands to the output of the Nth previous task. For eg if in the current task you want to check if the 3rd previous task had the word "started" in it's output, you could do something like "echo `@@AZ_TASKOUTPUT_PREVIOUS_3_NOSTRIP@@` — grep started" in the current task. Please note that this macro will only work for tasks that are at the same level i.e. this macro can only be used for tasks that share the same parent task. Also note that the entire output including newlines will be used, this is likely not what you want.

Using the macro `@@AZ_TASKOUTPUT_PREVIOUS_<N>_STRIP@@`. This macro works exactly like `@@AZ_TASKOUTPUT_PREVIOUS_<N>_NOSTRIP@@` but strips any leading and following newlines. This macro is most suitable to be used in a script/program.

Using the macro `@@AZ_TASKOUTPUT_<name>_NOSTRIP@@`. This macro expands to the output of the task named ¡name¿. Please note that if task copy files has executed multiple times in this workflow, this macro will get the output for the most recent/relevant execution.

Using the macro `@@AZ_TASKOUTPUT_<name>_STRIP@@`. This macro works exactly like `@@AZ_TASKOUTPUT_<name>@@` but strips any leading and following newlines.

Using the macro `@@AZ_TASKOUTPUTFILE_PREVIOUS_<N>@@`. This macro creates a temporary file containing the output of the Nth previous task. The macro expands to the name of the temporary file. This macro allows the current task to access the output of a previous task as a local file. For eg: if in the current ask you want to check if the 3rd previous task has the word "started" in it's output, you could do something like grep started `@@AZ_TASKOUTPUTFILE_PREVIOUS_3@@`. Please note that this macro will only work for tasks that are at the same level i.e. this macro can only be used for tasks that share the same parent task.

Using the macro `@@AZ_TASKOUTPUTFILE_<name>@@`. This macro works just like `@@AZ_TASKOUTPUTFILE_PREVIOUS_<N>@@` except the temporary file contains the output of the task named `<name>`.

4.2 Query Entity, Machines and Credentials

Listing macros can be used to query the relationship between Entities, Machines and Credentials as well as generate a list of available objects.

The simplest form of the listing macro is `@@AZ_LIST_Object1(criterion).get(Object2(attr))@@`. This macro filters Object1 against a criterion and returns all the associated instances of Object2. For eg:

`@@AZ_LIST_Entity(etype="DATABASE").get(Machine(name))@@` will return a list of comma separated names of each machine associated with a database

`@@AZ_LIST_Credential(uid="toor").get(Entity(id))@@` will return a list of comma separated ids of each entity for which your group has a credential with uid="toor"

`@@AZ_LIST_Machine(name="THIS").get(Property(name))@@` will return the value of @@name@@ if it is set on the current machine

The list of Objects that can be queried are:

- Entity
- Machine
- Credential
- Property

The list of attributes of an Entity that can be queried and filtered by table 4.1

Table 4.1: Attributes of Entity

Attribute	**Remark**
id	An integer uniquely identifying this Entity
etype	The type of this entity. One of database, webserver, ldapserver, and other
version	A version number, a string
name	The name of the entity
parent	The parent entity id

The list of attributes of a machine that can be queried and filtered by table 4.2

Table 4.2: Attributes of Machine

Attribute	**Remark**
id	An integer uniquely identifying this Machine
name	The name of the machine
address	The ip address or fqdn
os	The operating system on the machine

The list of attributes of a credential that can be queried and filtered by table 4.3

Table 4.3: Attributes of Credentials

Attribute	**Remark**
id	An integer uniquely identifying this Credential
name	The name of the credential
uid	The uid of the credential

Finally "name" is the attribute of a property that can be queried.

Note: For Entity, Machine and Credential, the special criterion name="THIS" signifies the current object.

`@@AZ_LIST_Entity(name="THIS").get(Machine(address))@@` will return the address of all machines associated with the Entity the in use by the current Task

Note: For For Entity, Machine, Credential, and property, the special criterion

name=”ALL” signifies all objects of this type.

`@@AZ_LIST_Entity(name="ALL").get(Machine(address))@@` will return the address of all machines associated with all Entities in the system.

More examples:

- `@@AZ_LIST_Entity(name="ALL").get(Entity(name))@@` will return the name of all the entities in the system
- `@@AZ_LIST_Machine(name="THIS").get(Credential(uid))@@` will return the uid of your credentials associated with the current machine
- `@@AZ_LIST_Credential(uid="turing").get(Credential(name))@@` will return the name of your credentials which have uid=turing

Specifying the delimiter The `@@AZ_LIST_Object1`(criterion).get(Object2(attr), delimiter=”delim”)@@ form specifies the delimiter to use while creating the output. The delimiter is always a double quoted string. The default delimiter is ”,”.

@@AZ_LIST_Entity(name=”ALL”).get(Machine(id), delimiter=”**”)@@ will return output in the form 1**2**3**4**5

@@AZ_LIST_Credential(name=”ALL”).get(Machine(address), delimiter=`"\n"`)@@ will return each machine address in a new line

Specifying the Object1 criterion

- Form : attribute=”value”

  We have already seen this form. An example:

  @@AZ_LIST_Entity(name=”core-router-1”).get(Credential(uid), delimiter=”:”)@@. The list of valid attributes that can be used for each object type is already given in the forementioned tables.

- Form: attribute ”value”

  This form of criterion treats value as if it were a property, seeks to expand it and then convert it to the attribute=”value” form. An example

  `@@AZ_LIST_Entity(name~"router_type")`.get(Credential(uid), delimiter=”:”)@@. `"router_type"` is treated as if it were @@router_type@@; once this is expanded the macro automaticatically converts to the @@AZ_LIST_Entity(name=”expanded_value”).get(Credential(uid), delimiter=”:”)@@

- Form: hasprop=”property”

  This criterion matches objects that have a property by the name ”property”. For eg @@AZ_LIST_Entity(hasprop=”restartable”).get(Machine(id))@@ will only get ids of machines that are associated with entities that have the property @@restartable@@ set. The value of @@restartable@@ is not important in this criterion.

- Form: hasprop~”property”

  This form will first expand @@property@@. The macro will then converts to the hasprop=”expanded_property” form.

- Form: "prop"="value"

  This criterion matches objects that have a @@prop@@=value set. This is a stricter variant of the hasprop="value" form, unlike which not only should @@prop@@ be present but must expand to "value". An example
  @@AZ_LIST_Machine("compress"="gzip").get(Credential(id))@@ will return the credential ids for machines that have a property "compress" set which expands to the value "gzip".

- Form: "prop"~"value"

  This criterion first expands value as if it were a property and then converts to the "prop"="expanded_value" form.

- Form: "~prop"="value" and "~prop"~"value"

  This critera first expands prop as if it were a property and then converts to one of the two forms "expanded_prop"="value" or "expanded_prop"~"value" **Note:** Property cannot be specified as the first Object

Specifying the Object2 filter criterion

- Form: get(Object2(attr))

  We have already seen this form.
  @@AZ_LIST_Entity(name="core-router-1").get(Credential(uid))@@. get(Credential(uid)) specifies that we are interested in the "uid" attribute of filtered Credentials. The list of applicable attributes has been forementioned.

- Form: get(Object2(attr1), attr2="value")

  This critera only matches Objects whose attr2=value.

  @@AZ_LIST_Entity(name="ftpd").get(Credential(name), uid="backup")@@ will only return names of Credentials that are associated with the "ftpd" Entity and have uid="backup".

- Form: get(Object2(attr1), attr2~"value")

  This criterion will expand "value" as if it were a property and then convert to the get(Object2(attr1), attr2="expanded_value") form

- Form: get(Object2(attr1), hasprop="prop")

  This criterion will only match objects that have a property called @@prop@@ set.

- Form: get(Object2(attr1), hasprop~"prop")

  This criterion will first expand prop and then convert to the

  get(Object2(attr1), hasprop="expanded_prop") form.

- Form: get(Object2(attr1), "prop"="val")

  This criterion will match objects that have a property @@prop@@ which expands to the value val

- Form: get(Object2(attr1), "prop"~"val")

  This criterion will expand val as if it were a property and convert to the get(Object2(attr1), "prop"="expanded_val") form

- Form: get(Object2(attr1), "prop"~"val")

  This criterion will expand val as if it were a property and convert to the get(Object2(attr1), "prop"="expanded_val") form

- Form: get(Object2(attr1), "~prop"="val") and get(Object2(attr1), " prop"~"val")

  This criterion will expand prop as if it were a property and convert to the get(Object2(attr1), "expanded_prop"="val") or get(Object2(attr1), "expanded_prop" "val") forms
  Note: Filter criteria cannot be applied to the Property object.

Note: If multiple filter critera are presented they are all anded together in the following order of descending precedence hasprops="prop", "prop"="value", attr="value".

Eg: @@AZ_LIST_Entity(name="ALL").get(Machine(address), hasprop="pingable", "~ip_type"="~selected_ip")@@ will first select machines with the @@pingable@@ property and then match those with "~ip_type"=" selected_ip" filter.

Applying offset & limit to returned values You can specify that only a subset of the final filtered output be returned. A single return value or a subset can be selected. The subset selection is performed by appending [a:b] to the macro where a and b are optional positive or negative bounding indexes.

For eg @@AZ_LIST_Entity(name="ALL").get(Entity(name))[2] gets only the 3rd entity name from the resulting list. @@AZ_LIST_Entity(name="ALL").get(Entity(name))[1:5]@@ gets 2,3,4 & 5th entity names (Like all self respecting computer scientists, we too count from zero).

The various forms of subset selection or slicing are:

- Form : [n] Select the nth value from the list. Eg [0] selects the 1st value, [3] selects the 4th value, [k] select the k+1th value

- Form : [-n] A negative index accesses elements from the end of the list counting backwards. The last element is always li[-1]

- Form : [a:b] A subset of values starting from a+1th to bth values. Eg [2:7] will return 5 values 2,3,4,5,6

- Form : [-a:b] & [a:-b] A subset of values starting from a+1th to bth values returning b - a values in total

- Form : [a:] & [:b] Implies [a:full length] & [0:b]

Interpreting a few complex macro examples

- @@AZ_LIST_Credential(name="ALL").get(Credential(name), delimiter=" — ", uid~"inp_id")[:-2]@@

  - Fetch all credentials beloning to my group
  - Expand @@inp_id@@
  - For each credential from (1) only select those whose uid matches the value in (2)
  - From the generated list select all values except the last 2
  - Concatenate all the selected values using " — " and return

- @@AZ_LIST_Machine("motoko"~"kusunagi", global).get(Entity(name, global), hasprop~"aramaki", ~"batou"~"saito", delimiter="—")[-5:-3]@@
  - Expand @@motoko@@ and @@kusunagi@@
  - Fetch all machines in the system that have a property-value matching expanded_motoko=expanded_kusunagi
  - Get all Entities associated with the machines from (2)
  - Expand @@aramaki@@, @@batou@@, @@saito@@
  - From the Entity list in (3), select a subset that have a property matching expanded_aramaki
  - From the Entity list in (5), select a subset that have a property-value matching expanded_batou=expanded_saito
  - From the Entity list in (6), drop the first 5 and the last 3
  - Concatenate the remaining Entities using "—" as a delimiter and return

4.3 Nested Macro Evaluation

There may be cases when a two level of macro evaluation is required. For example, we can use the call `@@AZ_GET_"user"="<user name>"_"email"@@` to fetch the email id of a epsilon user called `<user name>`. Suppose this user name is contained in a macro variable called "username" defined else where. To evaluate such a nested structure, we can use any one of the following methods

```
@@AZ_GET_"user"~"username"_"email"@@
@@AZ_GET_"user"="^^username^^"_"email"@@
@@(get-macro (concat AZ_GET_"user"=" (get-macro username) "_"email"))@@
```

In other examples of this nature,

- the output of a task whose name is residing in a variable @@taskname@@, can be fetched using `@@AZ_OUTPUT_"^^taskname^^"@@`.
- `@@AZ_REPORTABLE_"\%(^^output^^)s"_"^^mytags^^"@@` to report the outputs in format contained in variable "output" and with tags contained in variable "mytags".
- `@@AZ_LIST_Entity(name="^^ename^^").get(Machine(name))@@` can fetch the machines grouped together under entity in variable "ename"

Chapter 5

Epsilon API

Epsilon has given many utility APIs to ease the task of writing workflows. All the epsilon APIs are available when the task is of "Epsilon Script" type.

5.1 LDAP APIs for Microsoft AD Integration

Use these APIs for interaction with Microsoft Active Directory components and other LDAP services.

5.1.1 AD Classes

- a = ad(m=None, c=None, use_ssl=False, port=None, timeout=None, **attrs)

```
m - machine name
c - credential name
MC can be set via EMC pane also.

use_ssl = True  (ldaps is used and port = 636 by default)
        = False (ldap is used and port = 389 by default)

port (if the ports are other than the default ports)
timeout - request timeout (in secs).
```

- ADObject. Base class for all AD objects

```
obj1 = a.obj.get_by_dn(object_dn, objclass, attrlist=None)
obj1 = a.obj.get(base_dn, objclass, attr, value, attrlist=None)
These apis will always return a unique object. None if object is not found.
MULTIPLE_ENTRIES_EXCEPTION if multiple objects are found.

objlist = a.obj.query(base_dn, objclass, attr, value, attrlist=None)
This api will always return a list of objects. Empty list for no objects.

obj1 = a.obj.create(object_dn, **attrs)
This api will create a new object with distinguished name = object_dn
```

```
obj1 = obj1.update(add_attrs=None, replace_attrs=None)
After modify operation, this api internally fetches the modified object
using get_by_dn.

obj1 = obj1.refresh(base_dn)
Object is refreshed using objectGUID internally.

The object returned by all the apis has two attributes set. dn and attrs.
obj_returned.dn - string
obj_returned.attrs - dict of key-value pairs

obj1.delete()
```

- ADGroup (Inherits from ADObject)

```
grp1 = a.group.get_by_dn(object_dn, attrlist = None)
grp1 = a.group.get(base_dn, attr, value, attrlist = None)
grplist = a.group.query(base_dn, attr, value, attrlist = None)
grp1 = a.group.create(base_dn, name, group_scope = 2, group_type = 1,
                          **attrs)

group_scope
1 - domain local
2 - global (default)
3 - universal

group_type
1 - security (default)
2 - distribution

grp1 = grp1.update(add_attrs=None, replace_attrs=None)
grp1 = grp1.refresh(base_dn)
grp1.delete()

grp1 = grp1.add_member_by_dn(object_dn)
grp1 = grp1.add_member(adobject)
After add_member operation, these apis internally fetches the modified
object using get_by_dn.
```

- ADUser (Inherits from ADObject)

```
user1 = a.user.get_by_dn(object_dn, attrlist = None)
user1 = a.user.get(base_dn, attr, value, attrlist = None)
userlist = a.user.query(base_dn, attr, value, attrlist = None)

user1 = a.user.create(base_dn, first_name, logon_name, new_password, last_name=None,
        initials=None, full_name=None, must_change_pwd=True, pwd_never_expires=False,
         is_account_disabled=False, **attrs)

This api first creates a user, then resets the user password and
```

```
then enables account if needed.

user2 = a.user.create_user(base_dn, first_name, logon_name, last_name=None,
                initials=None, full_name=None, **attrs)
This api just creates a disabled user.

Rules:
display name = full_name = first_name if last_name = None and initials=None
and full_name=None
display name = full_name = first_name last_name if initials=None and full_name=None
display name = full_name = first_name initials. last_name if full_name=None
sAMAccountName = logon_name.split("@")[0]

user1 = user1.update(add_attrs=None, replace_attrs=None)
user1 = user1.refresh(base_dn)
user1.delete()

user1 = user1.disable_account()
user1 = user1.enable_account()
user1 = user1.reset_password(new_password, must_change_pwd=True)
user1 = user1.unlock_account()
user1 = user1.force_change_password()
These apis internally fetches the modified object using get_by_dn
after the desired operation is performed.

user1.is_account_locked()
user1.is_account_disabled()
These apis return True if the account is locked/disabled, False otherwise
```

- ADOU (Inherits from ADObject)

```
ou1 = a.ou.get_by_dn(object_dn, attrlist = None)
ou1 = a.ou.get(base_dn, attr, value, attrlist = None)
oulist = a.ou.query(base_dn, attr, value, attrlist = None)
ou1 = a.ou.create(base_dn, name, **attrs)

ou1 = ou1.update(add_attrs=None, replace_attrs=None)
ou1 = ou1.refresh(base_dn)
ou1.delete()
```

Note:

  - dn - distinguished name of the object
  - base_dn - level in the tree hierarchy under which all the operations will be performed.

Parameters that are calculated internally can be overridden by supplying the same in attrs (key-value pairs)

```
objectClass
sAMAccountName
groupType(ADGroup)
displayName(ADUser)
userAccountControl(ADUser)
```

PARAMETERS

attrlist - list of attributes to be fetched

GET_BY_DN

```
object_dn -- distinguished name of the object to be fetched.
objclass -- object class to which the object belongs to.
```

GET

```
base_dn -- distinguished name of the object under which the object needs
to be searched
attr -- attribute to be searched for
value -- value of the attribute
```

CREATE

```
base_dn -- distinguished name of the parent under which the object needs
to be created
attrs -- dict of key-value pairs
```

UPDATE

```
add_attrs -- dict of key-value pairs to be added
replace_attrs -- dict of key-value pairs to be replaced
```

REFRESH

```
base_dn -- distinguished name of the object under which the object needs to
be searched after refresh.
```

LIST OF EXCEPTION NAMES [All the exceptions can be accessed via ildap.exceptionname]

```
LDAPError
ADMINLIMIT_EXCEEDED
AFFECTS_MULTIPLE_DSAS
ALIAS_DEREF_PROBLEM
ALIAS_PROBLEM
ALREADY_EXISTS
AUTH_UNKNOWN
BUSY
CLIENT_LOOP
COMPARE_FALSE
COMPARE_TRUE
CONFIDENTIALITY_REQUIRED
CONNECT_ERROR
CONSTRAINT_VIOLATION
CONTROL_NOT_FOUND
DECODING_ERROR
ENCODING_ERROR
FILTER_ERROR
```

```
INAPPROPRIATE_AUTH
INAPPROPRIATE_MATCHING
INSUFFICIENT_ACCESS
INVALID_CREDENTIALS
INVALID_DN_SYNTAX
INVALID_SYNTAX
IS_LEAF
LOCAL_ERROR
LOOP_DETECT
MORE_RESULTS_TO_RETURN
NAMING_VIOLATION
NO_OBJECT_CLASS_MODS
NOT_ALLOWED_ON_NONLEAF
NOT_ALLOWED_ON_RDN
NOT_SUPPORTED
NO_MEMORY
NO_OBJECT_CLASS_MODS
NO_RESULTS_RETURNED
NO_SUCH_ATTRIBUTE
NO_SUCH_OBJECT
OBJECT_CLASS_VIOLATION
OPERATIONS_ERROR
OTHER
PARAM_ERROR
PARTIAL_RESULTS
PROTOCOL_ERROR
RESULTS_TOO_LARGE
SASL_BIND_IN_PROGRESS
SERVER_DOWN
SIZELIMIT_EXCEEDED
STRONG_AUTH_NOT_SUPPORTED
STRONG_AUTH_REQUIRED
TIMELIMIT_EXCEEDED
TIMEOUT
TYPE_OR_VALUE_EXISTS
UNAVAILABLE
UNAVAILABLE_CRITICAL_EXTENSION
UNDEFINED_TYPE
UNWILLING_TO_PERFORM
USER_CANCELLED

## EPSILON ERRORS
MULTIPLE_ENTRIES_EXCEPTION
```

5.2 List of LDAP APIs

1. `_ldap_reset_password(user_dn, newpass, mustchpwd=False, m=None, c=None, sslport=636, timeout=None)`
   Reset password of the user found via user_dn.

2. `_ldap_reset_password_with_query(base_dn, attr, value, newpass, mustchpwd=False, m=None, c=None, sslport=636, timeout=None)`
   Reset password of the user found via base_dn, attr, value

3. `_ldap_unlock_account(user_dn, m=None, c=None, use_ssl=True, port=None, timeout=None)`
   Unlock account of the user found via user_dn.

4. `_ldap_unlock_account_with_query(base_dn, attr, value, m=None, c=None, use_ssl=True, port=None, timeout=None)`
   Unlock account of the user found via base_dn, attr, value

5. `_ldap_force_change_pwd(user_dn, m=None, c=None, use_ssl=True, port=None)`
   Set the must change password on next logon to true.

6. `_ldap_force_change_pwd_with_query(base_dn, attr, value, m=None, c=None, use_ssl=True, port=None)`
   Set the must change password on next logon to true.

7. `_ldap_is_account_locked(base_dn, attr, value, m=None, c=None, use_ssl=True, port=None)`
   Fetches whether the user account is locked or not.

8. `_ldap_is_account_disabled(base_dn, attr, value, m=None, c=None, use_ssl=True, port=None)`
   Fetches whether the user account is disabled or not.

9. `_ldap_get_user_dn(base_dn, attr, value, m=None, c=None, use_ssl=True, port=None)`
   Fetches user's distinguished name.

10. `_get_gpo(base_dn, attr, value, m=None, c=None, use_ssl=True, port=None)`
    Fetches group policy object.

11. `_get_ou(base_dn, attr, value, m=None, c=None, use_ssl=True, port=None)`
    Fetches organizational unit.

12. `_link_gpo(obj_dn, gpo_dn, m=None, c=None, use_ssl=True, port=None, gpoptions=0, istop=False)`
    Links group policy object to OU/domain etc.

13. `_get_group(base_dn, attr, value, m=None, c=None, use_ssl=True, port=None)`
    Fetches group.

14. `_add_user_group(user_dn, group_dn, m=None, c=None, use_ssl=True, port=None)`
    Adds user to a group.

15. `_delete_user_group(user_dn, group_dn, m=None, c=None, use_ssl=True, port=None)`
    Deletes user from a group.

16. `_add_group(base_dn, name, m=None, c=None, use_ssl=True, port=None)`
    Add a new group

17. `_add_ou(base_dn, name, m=None, c=None, use_ssl=True, port=None)`
    Add a new organizational unit.

18. `_delete_group(group_dn, m=None, c=None, use_ssl=True, port=None)`
    Delete a group

19. `_delete_ou(ou_dn, m=None, c=None, use_ssl=True, port=None)`
    Delete an organizational unit.

20. `_unlink_gpo(obj_dn, gpo_dn, m=None, c=None, use_ssl=True, port=None, gpoptions=0, istop=False)`
    Unlinks group policy object from OU/domain etc.

21. `_get_manager_mail(user_dn, m=None, c=None, use_ssl=True, port=None, chkhier=False)`
    Fetches manager's email id. If chkhier = True, then the manager chain is traversed to find the manager's email id if manager's mail id is not found at the first level.

22. `_ldap_authenticate(remotelogin=None, remotepass=None, m=None, c=None, port=None, use_ssl=True)`
    Checks whether the username-password combination is correct.

23. `_ldap_get_user_attributes(base_dn, attr, value, attrlist=[], m=None, c=None, port=None, use_ssl=True)`
    Fetches user attributes. attrlist - list of attributes to be fetched.

24. `_ldap_get_account_status(base_dn, attr, value, m=None, c=None, use_ssl=True, port=None, timeout=None)`
    Fetches user account status.

25. `ad_delete_user(user_dn, m=None, c=None, use_ssl=True, port=None)`
    Delete a user.

Parameters:

dn - distinguished name (unique identifier throughout AD)

base_dn - base_dn of the domain/OU etc. The search will be done only under the base_dn hierarchy. attr, value - key-value pair of attribute and its value.

timeout - request timeout (in secs).

m - machine name

c - credential name

MC can be set via EMC pane also.

use_ssl = True (ldaps is used and port = 636 by default)
= False (ldap is used and port = 389 by default)

port (if the ports are other than the default ports)

Response:

```
(ecode, data, message) tuple
```

5.3 APIs for Resuming Workflows

The following eScript API calls can be used to model the service desk related approval processing.

```
epsilon = get_epsilon()

epsilon.approval.get_pending()

epsilon.approval.approve(runid, sid, authtoken, notes='')
epsilon.approval.reject(runid, sid, authtoken, notes='')
epsilon.approval.decide(runid, sid, authtoken, decision, notes='')
```

For example the following code

```
#script

ep = get_epsilon()

pending = ep.approval.get_pending()

for i, p in enumerate(pending):
  print "PROCESSING:", pformat(p)
  if i % 2 == 0:
    doit = ep.approval.approve(p['runid'], p['sid'], p['auth'])
    print "APPROVE:", pformat(doit)
  else:
    doit = ep.approval.reject(p['runid'], p['sid'], p['auth'])
    print "REJECT:", pformat(doit)
```

5.4 API to Remote Copy a File

To copy a file from one host to another over the network. Epsilon provides the following API. This API can extract the credentials from the Epsilon credentials, rather than need to pass the passwords as token.

```
_remote_copy(from_file, to_file,
                    from_mac=None, from_cred=None, to_mac=None, to_cred=None,
                    program="epsilon", options_to_program=None, timeout=None):
```

- from_file - the file to copy
- to_file - the file to create
- from_mac - the machine to copy from (defaults to task machine)
- from_cred - login for "from" machine (defaults to task cred)
- to_mac - the machine to copy to (defaults to task machine)
- to_cred - login for "to" machine (defaults to task cred)
- program - the program to use. 'epsilon' copies through epsilon using sftp, any other input is taken as the program to do direct transfer with, supports scp, rsync, and wmi Defaults to 'epsilon'
- options_to_program - passthru to the program

For example, the following code snippet can run in a task type run shell and script type, Epsilon Script

```
#script

result =  _remote_copy('/installer/epsilon.run', '/epsilon.run',
           from_mac='builder64', from_cred='build@builder64',
            to_mac='test64', to_cred='root@testvm',
               program='scp', options_to_program=None, timeout=600)

print result
```

5.5 JDBC APIs

Ask you Epsilon Administrator to install the jdbc driver for the database that you want to connect to using jdbc.

Sample JDBC URLs

```
dburl="jdbc:postgresql://localhost:5432/epsilon"
driver="org.postgresql.Driver"

dburl="jdbc:sqlserver://172.16.61.137:1433;database=testdb"
```

```
driver="com.microsoft.sqlserver.jdbc.SQLServerDriver"

dburl="jdbc:mysql://localhost:3306/epsilon"
driver="com.mysql.jdbc.Driver"

dburl="jdbc:oracle:thin:@172.16.61.137:1521:epsilon"
driver="oracle.jdbc.OracleDriver"
```

the list of JDBC APIs are give below JDBC APIS

1. **Common apis - session and standalone jdbc**

   (a) j1=jdbc(dburl=None, dbuser=None, dbpass=None, driver=None, validate=None, no_cache=None, session=None, isnew=None)

   dburl, dbuser, dbpass, driver are mandatory parameters for a new session/standalone object. [they can also be configured via task_props, jdbc_url, jdbc_driver, jdbc_user, jdbc_pass]

   validate and no_cache are optional params validate - if a different validation query needs to be used. We use "select 1 from dual" for oracle and "select 1" for other DBs if no_cache is there e.g no_cache=1, caching wont be used.

   DONT use session and isnew for now.

   if session and isnew are not none, a new session is created.

   if isnew is none and session is not none, earlier created session is reused.

   if isnew and session are none, jdbc object can be used to fire individual queries.

   (b) j1.execute(stmt=None, verb=None, tobuf=None, start=None, limit=None, fmt=None, rsname=None)

   stmt and verb('query, 'update) are mandatory parameters

   By default output will be printed to console. If tobuf is there e.g tobuf=1, then it can be stored in a variable. start = 1, limit = 100 by default. start and limit must be greater than 0.

   DONT use fmt and rsname for now. rsname can used for session requests to associate a resultset with a query executed. Then getrows and getnextrows can be used.

2. **Session specific apis**

   (a) j1.rows(start=None, limit=None, rsname=None, metadata=None, tobuf=None)

   get rows from offset=start and limit number of items

   rsname is mandatory.

   metadata - whether you want the metadata also.

   (b) j1.rowsnext(limit=None, rsname=None, metadata=None, tobuf=None)

   get limit number of items. offset is maintained at the server

   (c) j1.begin(tobuf=None)

   begin transaction. set autocommit to false.

   (d) j1.commit(tobuf=None)

   commit transaction

   (e) j1.rollback(tobuf=None)

   rollback transaction

(f) j1.close(tobuf=None)

close session. clears session key associated with connection and releases it to the pool.

Session apis can span across multiple tasks.

5.6 Date-Time Manipulation in Epsilon Scripts

Epsilon scripts are python codes. for security reasons, you can not import Python libraries inside an Epsilon scripts. But some libraries such as datetime can be accessible as _datetime.

For example the following script can be used to find the date, 7 days back, and output in the format 'DD-MON-YYYY' format.

```
#script

  print "from_date=%s"  % ((_datetime.datetime.today() -
   _datetime.timedelta(7)).strftime("%d-%b-%y") )
```

The rest of the datetime functionality can be got from any python reference manual.

Chapter 6

S-Expression Language

6.1 Introduction

Epsilon supports a S-expression language called Exp. The language is hosted via the macro system and is usable everywhere macros can be used.

Exp is modeled after the scheme family of languages, and is purely functional.

6.2 Syntax

Exp is accessed via the macro system with the following syntax:

```
@@( list-of-expressions )@@
```

$list - of - expressions$ is a list of one or more *expressions*. Each *expression* can itself be a $list - of - expressions$ surrounded by parenthesis (and).

Some example expressions:

```
@@( + 1 2 3 )@@
@@( + 1 ( * 4 5 ) 3 )@@
```

In the first example the expression is a list of 4 elements: +, 1, 2, 3.

In the second example the expression is a list of 4 elements, but the 3rd element is another list of 3 elements: $*$, 4, 5.

Lists can be nested up to any depth, limited by the memory on the server.

6.2.1 Data Types

Exp supports the following data types:

1. Number
2. String
3. List/Arrays
4. Functions
5. Regular Expressions

Number

A number is an integer or floating point number. Examples: 4, 3.1415

String

Strings are text tokens in matching double or single quotes. Both single ad double quotes within the string can be escaped with the character. The character can itself be escaped as
.

Lists are *polymorphic* and can contain objects of any of the types (including other lists).

6.2.2 Execution

The execution model followed by Exp is modeled after the Lisp family of languages. We talk about *evaluating* an *expression*, which means to find out what *value* that expression will compute to.

For evaluating a $list - of - expressions$ the first element of the list is used as a function which is passed all the other elements as arguments. Each *expression* in $list - of - expressions$ is itself evaluated until it results in a value.

A string or number is evaluated as itself. For example the following expression is evaluated in the following steps:

$list - of - expression$ can be composed of more nested expressions which are recursively evaluated.

6.3 Built In Operators

Exp ships with a number of built in operators. These operators are summarized in this section. Using the *lambda* operator you can define your own functions.

In all the examples the → symbol implies that the expression evaluates to the pointed result.

6.3.1 + (addition)

The + operator adds two or more numbers. The syntax is

```
(+ expression1 expression2 ...)
```

+ adds all the elements supplied to it. All the elements should be numbers.

```
(+ 1 2) → 3
(+ 1 2 5 9) → 17
(+ 1.5 2.5) → 4
```

6.3.2 - (subtraction)

The - operator subtracts one number from another. The syntax is

Listing 6.1: Subtraction syntax

```
(- expression1 expression2)
```

```
(- 2 1) → 1
(- 0 1) → -1
```

6.3.3 * (multiplication)

The * operator multiplies two or more numbers. The syntax is

```
(* expression1 expression2 ...)
```

```
(* 2 1) → 2
(* 1 2 3) → 6
```

6.3.4 / (division)

The / operator divides one number with another. The syntax is

```
(/ expression1 expression2)
```

```
(/ 2 1) → 2
(/ 1 2) → 0.5
```

6.3.5 > (greater than)

The > operator compares two numbers with each other. It returns `True` if the first number is greater than the second number, `False` otherwise. The syntax is

```
(> expression1 expression2)
```

```
(> 2 1) → True
(> 0 1) → False
```

6.3.6 < (less than)

The < operator compares two numbers with each other. It returns `True` if the first number is less than the second number, `False` otherwise. The syntax is

```
(< expression1 expression2)
```

```
(< 2 1) → False
(< 0 1) → True
```

6.3.7 >= (greater than or equal to)

The `>=` operator compares two numbers with each other. It returns `True` if the first number is greater than or equal to the second number, `False` otherwise. The syntax is

```
(>= expression1 expression2)
```

```
(>= 2 1) → True
(>= 2 2) → True
(>= 2 0) → False
```

6.3.8 <= (less than or equal to)

The `<=` operator compares two numbers with each other. It returns `True` if the first number is less than or equal to the second number, `False` otherwise. The syntax is

```
(<= expression1 expression2)
```

```
(<= 1 2) → True
(<= 1 1) → True
(<= 2 1) → False
```

6.3.9 = (equality)

The `=` operator compares two values with each other. It returns `True` if the first object is equal to the second object, `False` otherwise. Values can be numbers, strings, lists or any other data type in Exp. The Syntax is

```
(= expression1 expression2)
```

Listing 6.2: = (equality) example

```
(= 2 1) → False
(= 1 1) → True
(= "hello world" "hello world") → True
(= "abc" "def") → False
```

6.3.10 equal?

The `equal?` operator is a an alias for the `=` operator.

Listing 6.3: equal? example

```
(equal? 2 1) → False
(equal? 1 1) → True
(equal? "hello world" "hello world") → True
(equal? "abc" "def") → False
```

6.3.11 and?

The `and?` operator returns `True` if all the expressions supplied evaluate to `True`, otherwise returns `False`.

The Syntax is

```
(and? expression1 expression2 ...)
```

Listing 6.4: and? examples

```
(and? 1 2 3) → True
(and? 0 1 2) → False
(and? "") → False
(and? "hello world") → True
(and? 0 "" "hello world") → False
```

6.3.12 or?

The `or?` operator returns `True` if any one of the expressions supplied evaluate to `True`, otherwise returns `False`.

The Syntax is

```
(or? expression1 expression2 ...)
```

Listing 6.5: or? examples

```
(or? 1 2 3) → True
(or? 0 1 2) → True
(or? "") → False
(or? "hello world") → True
(or? 0 "" 0) → False
```

6.3.13 true?

The `true?` operator returns `True` if the expression supplied evaluates to `True`, otherwise returns `False`.

The Syntax is

```
(true? expression)
```

Listing 6.6: true? examples

```
(true? 1) → True
(true? 0) → False
(true? "") → False
(true? "hello world") → True
```

6.3.14 false?

The `false?` operator returns `True` if the expression supplied evaluates to `False`, otherwise returns `False`.

The Syntax is

```
(false? expression)
```

Listing 6.7: false? examples

```
(false? 1) → False
(false? 0) → True
(false? "") → True
(false? "hello world") → False
```

6.3.15 quote

The `quote` operator prevents the supplied expressions from being evaluated and returns them as is.

The Syntax is

```
(quote expression1 expression2 ...)
```

Listing 6.8: quote examples

```
(quote 1) → (1)
(quote "abc" "def") → ("abc" "def")
(quote + 1 2 3) → (+ 1 2 3)
```

6.3.16 list

The `list` operator constructs a list of all the supplied expressions

The Syntax is

```
(list expression1 expression2 ...)
```

Listing 6.9: list examples

```
(list 1) → (1)
(list "abc") → ("abc")
(list 1 2 3) → (1 2 3)
(list "hello world" "abc") → ("hello world" "abc")
(list 1 2 (* 3 0)) → (1 2 0)
```

6.3.17 list?

The `list?` operator returns `True` if the expression supplied is a list, otherwise returns `False`.

The Syntax is

```
(list? expression)
```

Listing 6.10: list? examples

```
(list? 1) → False
(list? "abc") → False
(list? (quote 1 2 3)) → True
(list? (quote "hello world" "abc")) → True
```

6.3.18 if

The `if` operator makes if-else choices. It takes three arguments. If the first argument (*expression1*) evaluates to `True` the second argument (*expression2*) is evaluated. Otherwise if the first argument evaluates to `False` the third argument (*expression3*) is evaluated.

The `if` operator can be summarized as: if the first expression is true then evaluate the second expression else evaluate the third expression.

The Syntax is

```
(if expression1 expression2 expression3)
```

Listing 6.11: if examples

```
(if 1 2 3) → 2
(if 0 2 3) → 3
(if (<= 1 2) 300 400) → 300
(if 0 "test was true" "test was false") → "test was false"

(if (and? (= 1 2) (= 2 2))
    "equal"
    "not equal") → "not equal"

(if (or? (= 1 2) (= 2 2))
    "equal"
    "not equal") → "equal"

(if (or? (= 1 2) (= 2 2))
    (* 100 2)
    0) → 200
```

6.3.19 define

The `define` operator sets a variable to the value of an expression.

The Syntax is

```
(define variable expression)
```

Listing 6.12: define examples

```
(define a 1)
(define my-name "epsilon")
(define c (quote 1 2 3))
(define my-list (list "hello world" "abc"))
(define my-sum (+ 1 2 3))

(define abs
      (lambda (x)
             (if (< x 0)
                 (- 0 x)
                 x)) → define the mathematical abs function
```

6.3.20 set!

The `set!` operator updates a variable to the value of an expression and does not return anything. `set!` can only update variables which have already been defined with `define`.

The Syntax is

```
(set! variable expression)
```

Listing 6.13: set! examples

```
(set! a 1)
(set! b "abc")
(set! c (quote 1 2 3))
(set! z (list "hello world" "abc"))
```

6.3.21 lambda

The `lambda` operator creates and returns a new anonymous function with the given list of arguments and the expression as function body.

The Syntax is

```
(lambda (variable-list) expression)
```

To assign the function to a name use `define` operator.

Listing 6.14: lambda examples

```
(lambda (x) x) → function which takes one argument x and returns it

(lambda (x y) (+ x y)) → function which takes two arguments x, y and
    returns their sum

(lambda (x)
       (if (< x 0)
           (- 0 x)
           x)) → function which takes one argument x and returns the
               absolute value of x
```

6.3.22 function

The `function` operator is an alias for `lambda`.

6.3.23 fn

The `fn` operator is an alias for `lambda`.

6.3.24 begin

The `begin` operator evaluates the supplied expressions and return the value of the last expression.

The Syntax is

```
(begin expression1 expression2 ...)
```

Listing 6.15: begin examples

```
(begin 1 2 3) → 3
(begin (+ 1 2) (* 1 2) (/ 1 2)) → 0.5
```

6.3.25 length

The `length` operator returns the number of a characters for string arguments, and the number of elements for list arguments.

The Syntax is

```
(length expression)
```

Listing 6.16: length examples

```
(length (list 1 2 3)) → 3
(length "Hello World") → 11
(length "") → 0
```

6.3.26 integer

The `integer` operator converts a string to an integer.

The Syntax is

```
(integer expression)
```

Listing 6.17: integer examples

```
(integer "1") → 1
(integer 55) → 55
(integer "145") → 145
```

6.3.27 number

The `number` operator converts a string or integer into a floating point number.

The Syntax is

```
(number expression)
```

Listing 6.18: number examples

```
(number "1.45") → 1.45
(number 0) → 0.0
(number "1") → 1.0
(number "-45") → -45.0
```

6.3.28 string

The `string` operator converts and returns any object type into a string.

The Syntax is

```
(string expression)
```

Listing 6.19: string examples

```
(string 1) → "1"
(string (list 1 2 3)) → "1 2 3"
(string 1.45) → "1.45"
(string "hello world") → "hello world"
```

6.3.29 split

The `split` operator takes two arguments and splits the first argument with the second expression as a delimiter. It returns the split results as a list. `split` is the inverse of `join`.

The Syntax is

```
(split expression1 expression2)
```

Listing 6.20: split examples

```
(split "Hello World" " ") → ("Hello" "World")
(split "1,2,3" ",") → ("1" "2" "3")
(split "1,,2,,3" ",,") → ("1" "2" "3")
```

6.3.30 join

The `join` operator takes two expressions, and concatenates the elements of second expression with first expression as a delimiter. The elements of second expression list are converted to strings if necessary. `join` is the inverse of `split`.

The Syntax is

```
(join expression1 expression2)
```

Listing 6.21: join examples

```
(join "," (list "1" "2" "3")) → "1,2,3"
(join "," (list 1 2 3)) → "1,2,3"
(join " " (list "a" "b" "c")) → "a b c"
```

6.3.31 concat

The `concat` operator return all it's arguments concatenated as one string.

The Syntax is

```
(concat expression1 expression2 ...)
```

Listing 6.22: concat examples

```
(concat "1" "2") → "12"
(concat 0 1 2 3) → "0123"
(concat "Hello" " " "World") → "Hello World"
```

6.3.32 first

The `first` operator returns the first element of the argument, which must be a list.

The Syntax is

```
(first list-expression)
```

Listing 6.23: first examples

```
(first (list 1 2 3)) → 1
(first (list "a" "b" "c")) → "a"
(first (list (list "first") "second" "third")) → ("first")
```

6.3.33 car

The `car` operator is an alias for the `first` operator.

6.3.34 rest

The `rest` operator returns all the elements *except* the first of a list.

The Syntax is

```
(rest list-expression)
```

Listing 6.24: rest examples

```
(rest (list 1 2 3)) → (2 3)
(rest (list "a" "b" "c")) → ("b" "c")
(rest (list (list "rest") "second" "third")) → ("second" "third")
```

6.3.35 cdr

The `cdr` operator is an alias for the `rest` operator.

6.3.36 range

The `range` operator generates and returns numeric sequences.

The Syntax is

```
(range end)
(range begin end)
(range begin end step)
```

Listing 6.25: range examples

```
(range 5)) → (0 1 2 3 4)
(range 1 5) → (1 2 3 4)
(range 1 10 2) → (1 3 5 7 9)
(range 0 10 2) → (0 2 4 6 8)
```

6.3.37 map

The `map` operator calls the first argument (which should be a function) with each element of second argument (which should be a list) and returns the function values as a list.

The Syntax is

```
(map function list-expression)
```

Listing 6.26: map examples

```
(map length (list "abc" "defghi" "jklmnopqr")) → (3 6 9)
(map number (list "1" "2" "3")) → (1.0 2.0 3.0)

(map
  (lambda (x) (+ 1 x))
  (list 1 2 3)
 ) → (2 3 4)

(map
```

```
  (lambda (x) (* x x))
  (list 1 2 3)
 ) → (1 4 9)
```

6.3.38 reduce

The `reduce` operator calls the first argument (which should be a function) with two element of second argument (which should be a list) and stores the results in an accumulator. It repeats this process until all the elements of the second argument have been processed.

The Syntax is

```
(reduce function list-expression)
```

Listing 6.27: reduce examples

```
(reduce
  concat
  (list "abc" "defghi" "jklmnopqr")
 ) → "abcdegfhijklmnopqr"
(reduce + (list 1 2 3)) → 6
(reduce * (list 1 2 3 4)) → 24
(reduce
  (lambda (x y) (concat (string x) "@" (string y)))
  (list 1 2 3)
 ) → "1@2@3"
```

6.3.39 regexp-search

The Syntax is

```
(regexp-search regexp string-to-search)
```

6.3.40 regexp-substitute

The Syntax is

```
(regexp-substitute regexp string-to-search replacement [count])
```

6.3.41 regexp-find-all

The Syntax is

```
(regexp-find-all regexp string-to-search)
```

6.3.42 regexp-match-group

The Syntax is

```
(regexp-match-group match-object [group-number])
```

6.3.43 regexp-match-groups

The Syntax is

```
(regexp-match-groups match-object)
```

6.3.44 get-property

The Syntax is

```
(get-property property-name)
```

6.3.45 get-macro

The Syntax is

```
(get-macro 'macro-name')
```

6.3.46 filter-macro

The Syntax is

```
(filter-macro macro-name regexp [group-number])
```

6.3.47 filter-macro-lines

The Syntax is

```
(filter-macro-lines macro-name regexp [group-number])
```

Chapter 7

Storage and Type Interface

7.1 Introduction

Epsilon support defining new types by combining the built in types. Types can be thought as of structures or record in other languages.

Using the type system, Storage provides named tables of types as a way to save and persist data. The data is available via an API in escript.

7.2 Basic Types

Epsilon has some basic inbuilt types. All other types are defined in terms of these types.

7.2.1 String

The String type stores a string of any length, limited by memory and disk space on the server.

7.2.2 Number

The Number type stores a 32 bit integer value.

7.2.3 Secret

The secret type can stores a string which is encrypted on-disk.

7.2.4 Date

The Date type stores a timestamp.

7.2.5 Boolean

The Boolean type stores a True or False value.

7.2.6 Float

The Float type stores a floating point value.

7.3 User Defined Types

User specific types can be defined by combining the basic types. We will explore an example before delving into the actual syntax.

Consider a workflow that accesses network devices like switches, routers, repeaters. All of these devices have the following common attributes

1. name of the device
2. description
3. priority
4. password
5. active flag
6. power consumption
7. installation date

Our workflow wants to store details and status of these devices in a persistent manner. We will begin by defining a new type which encompasses these details.

```
network_device_type:
  {
    name: String
    description: String
    priority: Number
    password: Secret
    active: Boolean
    power: Float
    installed_on: Date
  }
```

After we've defined network_device_type it is available for use in other types as well. For example if we have a box which includes 2 network devices in one chassis we can define it as:

```
big_chassis_type:
  {
    device1: network_device_type
    device2: network_device_type
  }
```

To create the type network_device_type in Epsilon, use the following script in a task of type Shell script, with script type as epsilon script.

```
#script
s = get_storage()
type_name='network_device_type'
t = s.types.search(type_name)
if not t:
        t = s.types.create(type_name, { 'name': ('string', {}),
```

```
                                        'description': ('string',{}),
                                        'priority': ('number',{}),
                                        'password': ('secret',{}),
                                        'active': ('boolean',{}),
                                        'power': ('float',{}),
                                        'installed_on': ('date',{})
                        })
        print "Created new type %s" % type_name
else:
        print 'type %s already exits' % type_name
```

Next, a table to store data, using the type defined above can be done as follows,

```
table = s.table.search(table_name)
if not table:
        table = s.table.create(table_name, type_name)
        print "Created new table %s using type %s" % (table_name, type_name)
else:
        table = table[0]
        print 'table %s already exits' % table_name
```

Insert a row into this table using,

```
print "creating row"
table.create_row({'name': "cisco 2600",
        'description': "Cisco router 2600",
        'priority': 10,
        'password': 'abc123',
        'active': True,
        'power': 100.2345})
print "created row"
```

The inserted row can be queried using,

```
x = table.get_rows()
for i in x:
        print i
```

For a filtered query use,

```
x = table.get_rows(criteria=[('name', '=', 'cisco 2600')])
```

Ordered by clause is

```
x = table.get_rows(orderby='.created')
```

Delete the data using

```
x = table.delete_row(criteria=[('name', '=', 'cisco 2600')])
```

Delete the table by

```
#script
s = get_storage()

table_name='network_device_details'
table = s.table.search(table_name)
table.delete()
```

To delete a type use

```
#script
s = get_storage()
type_name='network_device_type'
t = s.types.search(type_name)
t.delete()
```

Appendices

.1 Builtin Macro Listing

Macro Name	Explanation	Where to use
AZ_ID	The run id for this run	Generate a unique id per run of a workflow
AZ_RUN_ID	Same as @@AZ_ID@@	
AZ_RUN_LINK	Fully qualified http link to run status page for current run	Generate a fully qualified http link to run status page for current run
AZ_RUN_LINK_S	Fully qualified https link to run status page for current run	Generate a fully qualified https link to run status page for current run
AZ_GLOBAL_UNIQUE	Same as @@AZ_ID@@	
AZ_RANDOM	A random number	Generate a unique random number everytime
AZ_UNIQUE	Same as @@AZ_RANDOM@@	
AZ_TASK_NUM	Number of the currently executing task	Obtain the number of the currently executing task
AZ_TASK_NAME	Name of the currently executing task	Obtain the name of the currently executing task
AZ_TASK_ID	Unique Id of the currently executing task	Obtain the unique id of the currently executing task
AZ_PARENT_TASK_NUM	Number of the currently executing task's parent task	Obtain the number of the currently executing task's parent task
AZ_PARENT_TASK_NAME	Name of the currently executing task's parent task	Obtain the name of the currently executing task's parent task
AZ_PARENT_TASK_ID	Unique Id of the currently executing task's parent task	Obtain the unique id of the currently executing task's parent task
AZ_USER_NAME	Name of the user executing the workflow	
AZ_USER_EMAIL	Email of the user executing the workflow	
AZ_GROUP_NAME	Name of the group executing the workflow	This is the group that the logged in user has selected for this session
AZ_GROUP_EMAIL	Email of the group executing the workflow	This is the group that the logged in user has selected for this session
AZ_USER_MANAGER_NAME	Name of the user's manager	
AZ_USER_MANAGER_EMAIL	Email of the user's manager	
AZ_OWNER_GROUP_NAME	Name of the group owning this workflow	In Epsilon, Workflows are owned by Groups. This macro always expands to the name of the owner of the workflow irrespective of who is actually executing the workflow
AZ_OWNER_GROUP_EMAIL	Email of the group owning this workflow	In Epsilon, Workflows are owned by Groups. This macro always expands to the email address of the owner of the workflow irrespective of who is actually executing the workflow
AZ_REMOTE_USER_NAME	The userid that this task executes as on the remote/target machine	Workflow tasks are executed on target machines using the userid configued for that task. This macro expands to the configured userid
AZ_ENTITY_ID	The ID of the entity configured for this task	
AZ_ENTITY_NAME	The name of the entity configured for this task	
AZ_MACHINE_ID	The ID of the machine configured for this task	This is the remote/target machine on which this task is executing
AZ_MACHINE_NAME	The name of the machine configured for this task	This is the remote/target machine on which this task is executing
AZ_MACHINE_ADDRESS	The address of the machine configured for this task	This is DNS name or the IP address of the remote/target machine on which this task is executing
AZ_CREDENTIAL_NAME	The name of the credential configured for this task	This is name of the credential on which this task is executing
AZ_CREDENTIAL_UID	The User ID of the credential configured for this task	This is User ID of the credential on which this task is executing
AZ_CREDENTIAL_TYPE	The Type of the credential configured for this task	This is the Type of the credential on which this task is executing

Macro Name	Explanation	Where to use
AZ_NOOP	A numeric value that has special meaning for DECISION tasks	If a DECISION task exits with the exit code @@AZ_NOOP@@, the decision task will not execute any of its children. Workflow execution will proceed as if the DECISION task has succeeded. The hardcoded value of @@AZ_NOOP@@ is "242". Read the DECISION task documentation for more information on this macro.
AZ_SCHEDULED	=1, Indicates if the workflow run is through a scheduler	Used to identify if the workflow is run by a scheduler
AZ_RUNSHELL_SCHEME	0 Use Paramiko;1 Use libssh2;2 SSH CLI;3 SSH EXEC	Indicates the Type of runshell invoked by a runshell task. Can be set as a property
RUN_ON_LOCAL	1 Allows users to run commands on karan host;2 enables the running of karan as an agent on a target machine	Indicates if a powershell session is invoked locally or remotely by karan during windows automation.
AZ_TODAY	The current timestamp	This is current timestamp. This value is evaluated whenever the macro is encountered. An example expansion is 2009-08-12 13:39:08
AZ_NOW	Same as @@AZ_TODAY@@	
AZ_YEAR_YYYY	Current year in YYYY format	eg: 2009
AZ_YEAR_YY	Current year in YY format	eg: 09
AZ_MONTH_MM	Current month in MM format	eg: 08
AZ_MONTH_SHORT	Current month short name	eg: Aug
AZ_MONTH_LONG	Current month long name	eg: August
AZ_DAY_OF_MONTH	Numeric day of month	eg: 12
AZ_DAY_OF_MONTH	Numeric day of month	eg: 12
AZ_DAY_OF_YEAR	Numeric day of year	The day of year from 1 to 365, eg: 12
AZ_WEEKNUMBER	Numeric week of the year	The week of year from 1 to 54, eg: 32
AZ_WEEKNUMBER_ISO	ISO Numeric week of the year	The ISO week of year from 1 to 54, eg: 32
AZ_WEEKDAY_NUMBER	Numeric day of the week	eg: 3
AZ_WEEKDAY_NAME_SHORT	Day of the week in short form	eg: Wed
AZ_WEEKDAY_NAME_LONG	Day of the week in long form	eg: Wednesday
AZ_HOUR_AM_PM	AM or PM	Expands to the string AM or PM depending on the current time eg: PM
AZ_HOUR_12	Numeric day in 12hr format	eg: 01
AZ_HOUR_24	Numeric day in military/24hr format	eg: 13
AZ_MINUTE	Numeric Minute	eg: 39
AZ_SECOND	Numeric Seconds	eg: 42
AZ_TIME_CUSTOM-<format>	Current time in user specified format	Use to obtain the current time in a format of your choice. For eg @@AZ_TIME_CUSTOM-%d/%m/%y %H-%M-%S@@ will expand to 12/08/09 13-39-08, whereas @@AZ_TIME_CUSTOM-%I%p %A@@ will expand to 01PM Wednesday. The % specifiers are standard iso format specifiers
AZ_IS_WEEKDAY	Return '1' if the current day is a weekday (mon-fri)	Use to obtain a fast "truth" check if the current day is a regular weekday i.e., returns "true" if the current day is NOT saturday or sunday
AZ_IS_LONG_WEEKDAY	Return '1' if the current day is a long weekday (mon-sat)	Use to obtain a fast "truth" check if the current day is a long weekday i.e., returns "true" if the current day is NOT sunday
AZ_IS_WITHIN_<time1>_<time2>	Return '1' if the current time is within the supplied time1 & time2 range	eg: @@AZ_IS_WITHIN_0900_1730@@ will return "true" if the current time is between 9am and 5:30pm. Both "time1" and "time2" are inclusive

All the time based macros are based on local server timezone. To work in GMT/UTC/Zulu timezone just append _GMT to all the time based macros. For eg AZ_HOUR_24_GMT, AZ_TODAY_GMT, AZ_TIME_CUSTOM_GMT-$< format >$,AZ_IS_WITHIN_GMT_$< t1 >$_$< t2 >$ etc

Macro Name	Explanation	Where to use
All the time based macros use the current time for macro expansion. This timestamp will vary for each macro within the same workflow because it is freshly computed for each time based macro. While this works in most cases, for some slow or time sensitive workflows you may want to work with one single timestamp. In such cases you can use the "GLOBAL" variant of these time macros. All the time based macros can be used as AZ_GLOBAL_< *macroname* > - for eg; AZ_GLOBAL_HOUR_24, AZ_GLOBAL_TODAY_GMT, AZ_GLOBAL_TIME_CUSTOM-< *format* >, AZ_GLOBAL_IS_WITHIN_GMT_< *t1* >_< *t2* >. The "GLOBAL" variants always use the workflow "start time" instead of computing the current time for every macro. All "GLOBAL" macros thus use the same timestamp.		
`AZ_TIME_CONV^^infmt^^outfmt^^intime`	Convert intime from infmt to outfmt	`@@AZ_TIME_CONV^^%D^^%s^^01/01/10@@` will convert 01/01/10 to seconds from the unix epoch (1 Jan 1970). i.e. the macro will expand to 1282670836. Use this macro to convert time values to different formats. infmt and outfmt must be valid ISO time formats. "man strftime" on a unix/linux machine for a list of valid format specifiers.
AZ_RWI_TASK_< *taskname* >_WF_< *wfname* >_RUNID	The RUN-ID of remote workflow 'wf-name' run by RWI task 'taskname'	Use this macro to obtain the RUN-ID of a remote workflow run by a RWI task earlier in the workflow. For eg @@AZ_RWI_TASK_call_alpha_WF_alpha_RUNID@@ will expand as the run-id of the remote workflow 'alpha' run by RWI task by name 'call_alpha' earlier in the workflow
AZ_RWI_TASK_< *taskname* >_WF_< *wfname* >_STATUS	The status of remote workflow 'wf-name' fetched by RWI task 'taskname'	Use this macro to obtain the status of a remote workflow queried by RWI task. For eg @@AZ_RWI_TASK_get_alpha_status_WF_alpha_STATU will expand as the status of the remote workflow 'alpha' queried by RWI task by name 'get_alpha_status' earlier in the workflow
AZ_TASK_PREVIOUS_< *N* >_EXITCODE	The exit code of the Nth previous task	Use this macro to obtain the exit code of the Nth previous task. For eg @@AZ_TASK_PREVIOUS_1_EXITCODE@@ will expand as the exit code of the task that executed previous to the current task, whereas @@AZ_TASK_PREVIOUS_3_EXITCODE@@ will expand as the exit code of the 3rd previous task from the current one. Please note that this macro will only work for tasks that are at the same level i.e. this macro can only be used for tasks that share the same parent task. For a more generic macro see @@AZ_TASK_¡name¿_EXITCODE@@
AZ_TASK_< *name* >_EXITCODE	The exit code of the task named < *name* >	Use this macro to obtain the exit code of a specific task by its name. For eg @@AZ_TASK_copy files_EXITCODE@@ will expand as the exit code of the task copy files. Please note that if task copy files has executed multiple times in this workflow, this macro will get the exitcode for the most recent/relevant execution.

Macro Name	Explanation	Where to use
AZ_TASK_PREVIOUS_< N >_STATUS	The status of the Nth previous task	Use this macro to obtain the status of the Nth previous task. For eg @@AZ_TASK_PREVIOUS_1_STATUS@@ will expand as the status of the task that executed previous to the current task, whereas @@AZ_TASK_PREVIOUS_3_STATUS@@ will expand as the status of the 3rd previous task from the current one. Please note that this macro will only work for tasks that are at the same level i.e. this macro can only be used for tasks that share the same parent task. For a more generic macro see @@AZ_TASK_< $name$ >_STATUS@@
AZ_TASK_< $name$ >_STATUS	The status of the task named < $name$ >	Use this macro to obtain the status of a specific task by its name. For eg @@AZ_TASK_copy files_STATUS@@ will expand as the status of the task copy files. Please note that if task copy files has executed multiple times in this workflow, this macro will get the status for the most recent/relevant execution. The most common status values are SUCCESS & FAILURE.
@@AZ_TASK_TIMEOUT@@	Time at which the current task will timeout.	Use this macro to obtain the time at which the current task will timeout. The time is returned in YYYY-MM-DD HH:MM:SS format.
@@AZ_APPROVAL_STATUS_ID@@	The status id for current approval task.	Generate a unique id for current approval task.
@@AZ_APPROVAL_URL_TOKEN@@	URL token for current approval task.	Generate a URL token for current approval task. This token can be used only in approval links.
@@AZ_APPROVAL_AUTH_TOKEN@@	Auth token for current approval task.	Generate an auth token for current approval task.
@@AZ_APPROVAL_URL@@	Fully qualified http link for current approval task.	Generate a fully qualified http link for current approval task. This URL resolves to approval decision page.
@@AZ_APPROVAL_URL_S@@	Fully qualified https link for current approval task.	Generate a fully qualified https link for current approval task. This URL resolves to approval decision page.
@@AZ_APPROVAL_TARGET@@	Target at which the current approval is targeted.	Returns target at which the current approval is targeted. Target can be user/-group name or email id.
@@AZ_MANUAL_STATUS_ID@@	The status id for current manual task.	Generate a unique id for current manual task.
@@AZ_MANUAL_URL@@	Fully qualified http link for current manual task.	Generate a fully qualified http link for current manual task. This URL resolves to manual decision page.
@@AZ_MANUAL_URL_S@@	Fully qualified https link for current manual task.	Generate a fully qualified https link for current manual task. This URL resolves to manual decision page.
@@AZ_MANUAL_TARGET@@	Target at which the current manual is targeted.	Returns target at which the current manual is targeted. Target can be user/group name or email id.

Macro Name	Explanation	Where to use
AZ_OUTPUT_"NAME"[_pat="pattern"][a:b:c]	Apply a filter pattern to the output of task "NAME" and return the ath:bth slice from the result	Use this macro to filter the output of a previously run task and return a subset of the matched lines. Both filtering and subset selection are optional. In it's simplest form @@AZ_OUTPUT_"NAME"@@ will return the output of the task "NAME". Similarily @@AZ_OUTPUT_"PREVIOUS_N"@@ will return the output of the Nth previously run sibling task. The more advanced form @@AZ_OUTPUT_"NAME"_pat="âlpha"@@ for example will return lines starting with the string "alpha". @@AZ_OUTPUT_"NAME"_pat="10\stimes"[1]@@ will return the 2nd line of the output from "NAME" starting with "10 times". An index of -1 will return the last line. An index of 3:7 will return the lines from 3 to 7. Please note that the special NAME "THIS" is reserved for the currently running task.
AZ_TASKOUTPUT_PREVIOUS_< N >	The output of the Nth previous task	Use this macro to obtain the output of the Nth previous task. For eg @@AZ_TASKOUTPUT_PREVIOUS_1@@ will expand as the entire output of the previous task. Please note that this macro will only work for tasks that are at the same level i.e. this macro can only be used for tasks that share the same parent task. Also note that the entire output including newlines will be used, this is likely not what you want. For a newline safe version see @@AZ_TASKOUTPUT_PREVIOUS_< N >_STRIP@@. For a more generic version of this macro see @@AZ_TASKOUTPUT_¡name¿@@
AZ_TASKOUTPUT_PREVIOUS_< N >_STRIP	The newline stripped output of the Nth previous task	Use this macro to obtain the newline stripped output of the Nth previous task. For eg @@AZ_TASKOUTPUT_PREVIOUS_1@@ will expand as the entire output of the previous task but without the starting and ending newlines. This form of output is more suitable to be included as part of another script/program. Please note that this macro will only work for tasks that are at the same level i.e. this macro can only be used for tasks that share the same parent task. For a more generic version of this macro see @@AZ_TASKOUTPUT_< $name$ >_STRIP@@
AZ_TASKOUTPUT_< $name$ >	The output of the task named < $name$ >	Use this macro to obtain the output of a specific task by its name. For eg @@AZ_TASKOUTPUT_copy files@@ will expand as the output of the task copy files. The entire output including starting and ending newlines will be used, this is likely not what you want. For a newline safe version see @@AZ_TASKOUTPUT_< $name$ >_STRIP@@ Please note that if task copy files has executed multiple times in this workflow, this macro will get the output for the most recent/relevant execution.

Macro Name	Explanation	Where to use
AZ_TASKOUTPUT_< *name* >_STRIP	The stripped output of the task named < *name* >	Use this macro to obtain the newline stripped output of a specific task by its name. For eg @@AZ_TASKOUTPUT_copy files_STRIP@@ will expand as the output of the task copy files but without the starting and ending newlines. This form of output is more suitable to be included as part of another script/program. Please note that if task copy files has executed multiple times in this workflow, this macro will get the output for the most recent/relevant execution.
AZ_TASKOUTPUTFILE_PREVIOUS_< *N* >	The name of a temporary file containing the output of the Nth previous task	When Epsilon encounters this macro, it creates a temporary file containing the entire output of the Nth previous task and expands this macro to the name of that file. This temporary file is created on the same target/remote machine that the current task is executing on. This allows the current task to access the output of a previous task from a file. For eg @@AZ_TASKOUTPUTFILE_PREVIOUS_1@@ will create a temporary file with the output of the previous task and expand this macro to the name of the temporary file. Please note that this macro will only work for tasks that are at the same level i.e. this macro can only be used for tasks that share the same parent task. For a more generic version of this macro see @@AZ_TASKOUTPUTFILE_< *name* >@@
AZ_TASKOUTPUTFILE_< *name* >	The name of a temporary file containing the output of the task named < *name* >	When Epsilon encounters this macro, it creates a temporary file containing the entire output of the task named ¡name¿ and expands this macro to the name of that file. This temporary file is created on the same target/remote machine that the current task is executing on. This allows the current task to access the output of a previous task from a file. For eg @@AZ_TASKOUTPUTFILE_copy files@@ will create a temporary file with the output of the copy files task and expand this macro to the name of the temporary file. Please note that if task copy files has executed multiple times in this workflow, this macro will get the output for the most recent/relevant execution.
AZ_REPORT_"name"_"fmt"	Extract information about the given task in the format provided	name is the name of a previously run task. fmt is any string. The macro returns this string after expanding any format specifiers it finds. A " inside the format string can be escaped by prefixing it with a \. The list of recognized format specifiers are: %(name)s Name of the task %(row_number)s Row number of macro output

Macro Name	Explanation	Where to use
		%(status)s Status of the task %(reason)s Reason for success/failure (if any specified) %(exit_code)d Exit code of task %(started)s Task start time %(finished)s Task finish time %(time)s Time taken by task execution %(time_human)s Time taken by task execution in human readable format %(runid)s Task runid %(comment)s Task comment %(type)s Task type %(output)s Task Output %(output_strip)s Task Output stripped of beginning and ending spaces and newlines %(output_html)s Task Output stripped of beginning and ending spaces and newlines and HTML escaped %(user.id)s Id of the user running the workflow %(user.name)s Name of the user running the workflow %(user.fullname)s Full name of the user running the workflow %(user.email)s Email of the user running the workflow %(group.id)s Id of the user running the workflow %(group.name)s Name of the group running the workflow %(group.email)s Email of the group running the workflow %(entity.id)s Id of the entity the task used %(entity.name)s Name of the entity the task used %(entity.etype)s Type of the entity the task used %(entity.version)s Version of the entity the task used %(machine.id)s Id of the machine the task used %(machine.name)s Name of the machine the task used %(machine.address)s Address of the machine the task used %(machine.ssh_port)s Port of the machine the task used %(machine.os)s OS of the machine the task used %(machine.owner)s Name of the group that owns the machine the task used

Macro Name	Explanation	Where to use
		%(credential.id)s Id of the credential the task used %(credential.name)s Name of the credential the task used %(credential.uid)s Username of the credential the task used %(credential.type)s Type of the credential the task used %(tags)s List of user defined reportable tags the task used %(tags:user)s List of user defined reportable tags the task used %(tags:system)s List of system defined reportable tags the task used %(tags:all)s List of all reportable tags the task used Example: @@AZ_REPORT_"start db"_"DB startup \"%(st)s\" with exitcode %(ex)d. Reason was : %(re)s"@@ may expand as DB startup "FAILURE" with exitcode 1. Reason was : msg libraries are missing
AZ_REPORT_"name"_TREE_"fmt"	Extract information about the given task in the format provided. Look in the task subtree for a success/failure reason	Same as @@AZ_REPORT_¡name¿_"fmt"@@ except the macro will examine all the children of name till it finds a child which has a success/failure reason set
AZ_REPORTABLE_"fmt"_"tags"	Extract information about all the tasks with Reportable flag set and all the optional tags used, in the format provided. The output will be returned as newline separated list of fmt.	Same as @@AZ_REPORT_¡name¿_"fmt"@@ except the macro will examine all the children of a workflow with Reportable flag set as true.
AZ_TOP_WORKFLOW_"fmt"	Extract information about the workflow which started this run.	Format specifiers are same as @@AZ_REPORT_¡name¿_"fmt"@@ except the macro will examine only the top level Workflow. For example, AZ_TOP_WORKFLOW_"workflow %(name)s with run id '%(runid)s'" expands as workflow shutdown_db with run id 1292341164824
AZ_UPDATE_"name"_"attr"="value"	Update the history attribute of a previously run task	Use this macro to update the history of a previously run task. name is the name of a previously run task. attr is the history attribute to be updated. Currently only status can be updated. value the new value of the attr. @@AZ_UPDATE_"diff files"_"status"="FAILURE"@@ will update the status of "diff files" task to FAILURE.
AZ_TREE_UPDATE_"name"_"attr"="value"	Update the history attribute of a previously run task and all its children recursively	Use this macro to update the history of a previously run task and all its children recursively. name is the name of a previously run task.

Macro Name	Explanation	Where to use
		attr is the history attribute to be updated. Currently only status can be updated. value the new value of the attr. @@AZ_TREE_UPDATE_"gather information"_"status"="FAILURE"@@ will update the status of "gather information" task and all its children to FAILURE.
AZ_GET_"oclass"exp"name"_"attr"	Get a attr value for specified object	Fetch specified attr for a object of type oclass. oclass specifies the type of object. Supported oclass: user, group exp specifies the query mechanism. '=' interprets name as name of object. '~' interprets name as name of property containing object name name implies object name if exp is =, property name which contains the object name. "THIS" implies current user. attr specifies object attribute to fetch. Supported attr per oclass: oclass user: fullname, email, manager, manager_email, manager_fullname, reportees oclass group: owner, owner_email, owner_fullname, members, children @@AZ_GET_"user"="aalok"_"manager_email"@@ fetches user aalok's manager's email @@AZ_GET_"user"="THIS"_"email"@@ fetches email of user running the workflow @@AZ_GET_"user"~"username"_"fullname"@@ queries user by name contained in property username. user's fullname is returned. @@AZ_GET_"Group"="wheel"_"members"@@ returns comma separted login names of all members of group wheel This macro raises an exception if it cannot find object or if unsupported attr is supplied.
AZ_PROPERTYFILE_< *propertyname* >	Access a file uploaded via the Input task (See the section for Input task)	This macro works like the AZ_TASKOUTPUTFILE_ macros in providing a temporary file on machine running the current task. The property_name is defined by the Input task. property_name is the name of the property used to upload the file.
AZ_PROPERTYFILENAME_< *propertyname* >	Access the name of a file uploaded via the Input task (See the section for Input task)	This macro works like the AZ_PROPERTYFILE_< macro in providing the original name of the uploaded file. The property_name is defined by the Input task.

Macro Name	Explanation	Where to use
		property_name is the name of the property used to upload the file.
AZ_COMBINED_LOGS_XML	Only for Email Task attachments: mail all the consolidated logs in XML format as an attachment	This macro when used as a file attachment in EMAIL task will mail all the consolidated logs (at the time the email task is run) as an attachment.
AZ_COMBINED_LOGS_TXT	Only for Email Task attachments: mail all the consolidated logs in Text format as an attachment	This macro when used as a file attachment in EMAIL task will mail all the consolidated logs (at the time the email task is run) as an attachment.